NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DESIGN OF A PROTOTYPE AUTONOMOUS AMPHIBIOUS WHEGS™ ROBOT FOR SURF-ZONE OPERATIONS

by

Jason L. Ward

June 2005

Thesis Advisor: Richard Harkins
Co-Advisor: Ravi Vaidyanathan

Approved for public release: distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Design of a Prototype autonomous Amphibious WHEGS™ Robot for Surf-Zone Operations		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Jason Ward			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

The Small Robot Initiative at the Naval Postgraduate School (NPS) has spent several years in development based on the Foster Miller lemmings platform. This platform, in conjunction with a commercial-off-the-shelf (COTS) control architecture, is capable of autonomous, land based waypoint navigation, self orientation, and rudimentary obstacle avoidance. It can receive waypoint information, manual control input, and transmit video and audio information back to a control station via 802.11 wireless communication. The introduction of the WHEGS™ design, developed at Case Western Reserve University, and a modified version of the COTS control system will provide a platform with greater speed, mobility and versatility. This thesis developed a prototype WHEGS™ vehicle and integrated the control system with improvements in the navigation routine through the addition of a dead reckoning sensor and calculation function. Although the mechanical design proved to be highly inefficient and unable to propel itself, the control system was successful, allowing integration with a more robust mechanical design from Case Western Reserve University. Follow on development and research will lighten the body through the use of carbon fiber and test the robots ability to maneuver effectively in the surf-zone.

14. SUBJECT TERMS autonomous, amphibious, WHEGS, robot			**15. NUMBER OF PAGES** 105
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DESIGN OF A PROTOTYPE AUTONOMOUS AMPHIBIOUS WHEGS™ ROBOT FOR SURF-ZONE OPERATIONS

Jason L. Ward
Lieutenant, United States Navy
B.S., United States Naval Academy, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2005

Author: Jason L. Ward

Approved by: Richard Harkins
Thesis Advisor

Ravi Vaidyanathan
Co-Advisor

James Luscombe
Chairman, Department of Applied Physics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Small Robot Initiative at the Naval Postgraduate School (NPS) has spent several years in development based on the Foster Miller lemmings platform. This platform, in conjunction with a commercial-off-the-shelf (COTS) control architecture, is capable of autonomous, land based waypoint navigation, self orientation, and rudimentary obstacle avoidance. It can receive waypoint information, manual control input, and transmit video and audio information back to a control station via 802.11 wireless communication. The introduction of the WHEGS™ design, developed at Case Western Reserve University, and a modified version of the COTS control system will provide a platform with greater speed, mobility and versatility. This thesis developed a prototype WHEGS™ vehicle and integrated the control system with improvements in the navigation routine through the addition of a dead reckoning sensor and calculation function. Although the mechanical design proved to be highly inefficient and unable to propel itself, the control system was successful, allowing integration with a more robust mechanical design from Case Western Reserve University. Follow on development and research will lighten the body through the use of carbon fiber and test the robots ability to maneuver effectively in the surf-zone.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my thesis advisor for the steady, continued support throughout my time at NPS, Ravi Vaidyanathan for the energy and vision that he and the WHEGS™ concept provided to me, the robotics team at NPS, John Clark in the Computer Science department for helping me decipher a sometimes foreign language, and most importantly my parents who have always supported me and publicly hide their shock that I've made it this far.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. NPS SMART DEVELOPMENT

The Naval Postgraduate School's Small Robotic Technology (SMART) initiative has been an ongoing research effort within the Combat Systems Science and Technology curriculum. The overarching concept of the SMART initiative is to develop applications of robotics for military use. The initial platform consisted of a Foster-Miller Lemmings platform that had been modified to accommodate Commercial-Off-The-Shelf (COTS) components to serve as the control and sensors arrays as well as increase the overall payload for ease of experimentation. This platform, affectionately named Bender© after the character by Matt Groening, shown in Figure 1, is designed with the idea of being able to host variable sensor packages on a platform that is semi-autonomous and capable of waypoint navigation, obstacle avoidance and basic dead reckoning.

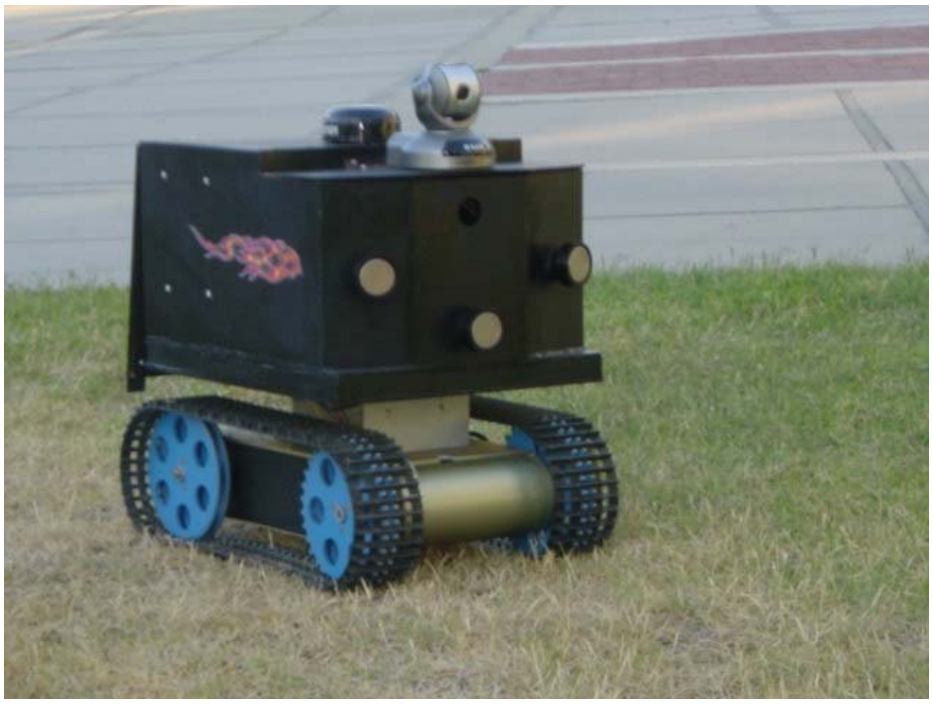

Figure 1. Bender©

The control interface is accomplished through the use of a JAVA application that provides GPS, compass and waypoint data, a scaled vector map, manual control buttons, tabs to view camera data and a communications window

to aid in monitoring Bender's actions and status. Bender© has gone through numerous modifications since the initial work in 2001, but the mission remains the same. Now, with the incorporation of work done at Case Western Reserve University, the SMART initiative will be taking a fairly significant step towards increased mobility while maintaining the autonomy and ease of interface completed to date.

B. WHEGS™ DEVELOPMENT FOR ADVANCED MOBILITY

There is a significant amount of interest in autonomous, amphibious robots capable of operating in the surf zone. Such robots would be capable of preparing amphibious landing zones by locating, classifying and mapping potential threats to a landing force such as obstacles and mines as well as performing general reconnaissance of the beachhead. In order to accomplish this, the vehicle would have to be rugged, highly mobile and capable of navigation in possibly extreme environments. In order to achieve the level of mobility required, recent research has focused on the use of legged or crawling platforms.

1. Biologically Inspired Mobility

Despite the advantages of wheeled (simple, efficient) and tracked (more mobile) running gears, their respective lack of terrain adaptability render them inappropriate for the proposed innovation. Although animal-like legged vehicles provide the most potential for locomotion over uneven substrates, their technological maturity is not such that they may be implemented in the near-term for field use. In this work, these issues will be surmounted by implementing the patented Whegs™ [Ref 1] wheel-legs running gear to capture the efficiency/ruggedness of wheeled/tracked vehicles and the advanced mobility of legged platforms.

In the Whegs™ concept [Ref 1], the basic walking principles of the cockroach were distilled out such that a relatively simple platform could be designed for mobility over harsh terrain. In particular, biologically based studies on the locomotion of cockroaches [Ref 2] have shown these creatures to be highly mobile and adaptable to a wide range of terrains. The cockroach has six

legs that are controlled differently depending on the terrain to be covered. Fundamental aspects of cockroach locomotion over complex terrain were identified where the animal could outperform the existing robot designs. By examining movement in these situations, the strategies that the animal used were identified and implemented as mechanical solutions in the robot. For these robots, the desired mobility capability was not addressed in the same manner as in the cockroach. Rather, the animal's mechanisms were abstracted such that their salient characteristics could be captured using more standard technology.

The nominal locomotion pattern found in cockroaches and most other insects is the tripod gait in which the front and rear legs on one side move in approximate synchrony with the middle leg on the opposite side. This tripod alternates with a second tripod made up of the remaining legs. Each leg moves rapidly through the swing phase and then slows down as it enters the stance phase and pushes the body forward. The front legs move through an arc on the order of the animal's height in order to reach the top of small barriers without the need for any change in leg movement [Ref 2]. Figure 2 specifically illustrates this pattern. The highlighted legs push forward while the remaining legs are lifted and moving forward preparing for the next step, then the highlighted legs lift and move forward while the remaining legs are now used to push.

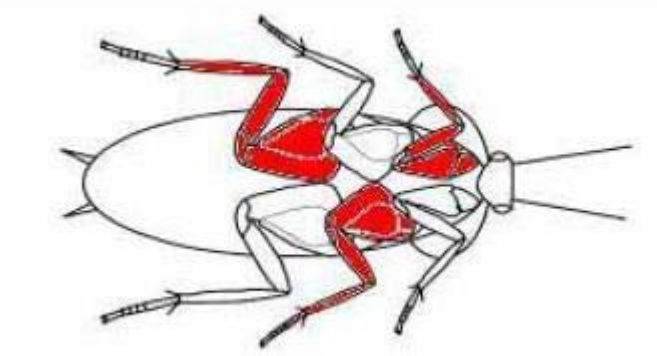

Figure 2. Cockroach using Tripod Gait [From Ref [3]

When the insect encounters an obstacle, as in Figure 3, the front legs of the insect move in unison while its body bends in the middle, allowing the front of the body to pitch up and climb higher. As it overcomes the obstacle, each pair of

legs moves in unison until they are over the obstacle and the body pitches down to prevent high-centering. Once the obstacle is passed, the legs return to the tripod gait.

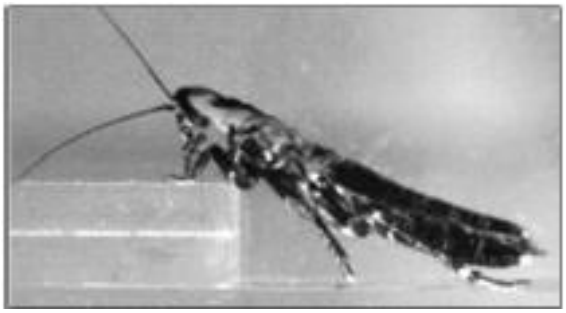

Figure 3. Cockroach climbing obstacle [From Ref. [3]

The Whegs™ design [Ref 1] incorporates the six-legged locomotion (two wheel-legs on each of three axles) as well as compliance in the wheel-legs to enable the platform to passively go in and out of the tripod gait through the use of a modified limited slip differential. Figure 4 shows how a single axle responds to an obstacle.

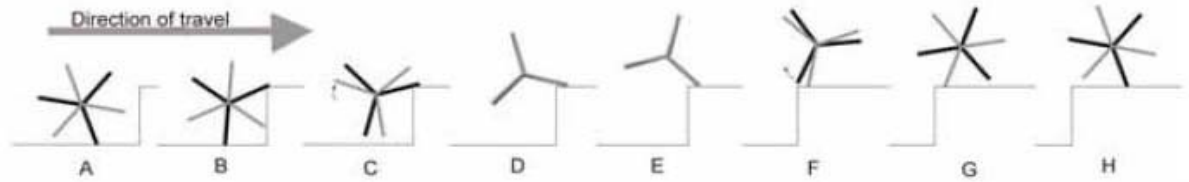

Figure 4. Compliance in Whegs™ design [From Ref. [3]

During normal travel, the left and right wheel-legs (shown as black and gray, respectively) are 60 degrees out of phase. As the axle strikes the obstacle, the left wheel-leg is applying most of the force to climb up, while at the same time, due to the compliance that is designed in, is rotating until it is in phase with the right wheel-leg and the force of lifting the axle over the obstacle is distributed for even support and maximum lifting power. Once the obstacle is overcome, the left wheel-leg quickly rotates back into phase allowing the platform to resume its tripod gait. Figures 8 and 9 (on page 13) show the three axle configuration, with the latter more clearly showing how the front and rear wheel-legs on one side are in phase with the middle wheel-leg of the other side. This provides three-point stability when the platform is running normally on an obstacle-free path.

II. AUTONOMOUS BEHAVIOR

A. CONCEPT OF AUTONOMY

In order to operate effectively in the surf zone, an autonomous platform will need to perform several functions. It will need to locate its position, maneuver around the environment and respond to cooperative players and commands. Determining position and maneuvering within the environment requires some form of navigation and control. In addition to basic navigation, the platform must be able to adapt to obstacles. Responding to commands and coordinating with other platforms requires a robust communication capability.

1. Navigation

Navigation of an autonomous platform around a work space can be accomplished through a variety of methods. In controlled environments, a platform can determine its position relative to one or more known fixed positions through range and/or bearing detection or triangulation using sonar, lights, radio beacons or radar. This method is fine for controlled or limited space environments but can be difficult to use in large scale workspaces.

In order to operate in the world at large, the platform can still determine its position relative to known objects, such as buildings or geographic landmarks, but that would require the ability to recognize the object as well as maintain a database of landmarks with their associated positions. The easier alternative is to use a global network of transmitters such as LORAN or GPS which are specifically designed for navigation. Both LORAN and GPS provide distributed transmitters that allow a receiver to pick up multiple signals which the receiver can used to triangulate its position. Advances in GPS technology, wider coverage, improved accuracy, low cost, and ease of integration make GPS preferable. Another method available is dead reckoning where the platform calculates its position from a known position through the use of shaft encoders, acoustic or laser range detectors, gyros and rate gyros or accelerometers. Shaft encoders are only useful if the drive train is in direct contact with the ground. Laser range finders are of limited or no use underwater and acoustic modems

can be challenging to integrate and calibrate. Accelerometers are not limited by the environment (provided they are sufficiently packaged) and through the use of integration hardware or software can provide velocity or distance information to the navigation routine.

2. Obstacle Avoidance

In a controlled environment, not much concern is given to obstacles as they are typically infrequent by design, and when they do arise, the robot can be stopped while the obstacle is removed. When operating in an uncontrolled environment, the robot must be designed to handle as many of the expected obstacles as possible within engineering limitations. Obstacles can be handled either passively or actively. Passive obstacle avoidance basically means being able to drive over, through, or bounce off of anything that lies in the path. Active obstacle avoidance requires that the platform first detect the obstacle and make a change in the navigation routine to steer around it. Detecting the obstacle can be accomplished with acoustic, laser, or contact sensors or a combination of these.

3. Communication Protocols

Although a truly autonomous vehicle, once programmed, should be able to operate without intervention, and as such do not need to communicate, most are not designed that way. Instead, they are designed to be semi-autonomous, return status or environmental data, or to cooperate with other robots. To accomplish these requirements, the platform must use some form of wireless transceiver, usually in the form of a modem or satellite communication. The simplest is a wireless 802.11 modem. From here, the modems can send data back and forth using either Transport Control Protocol (TCP) or User Datagram Protocol (UDP). UDP is essentially a connectionless communication scheme that allows broadcasting information with limited means of ensuring that the data is received correctly. TCP is connection oriented and has the ability to detect errors or lost data and trigger retransmission, but can be limited by the requirement for a connection.

B. ROBOT CONTROL

1. Waypoint Handling

In order for the autonomous vehicle to navigate, it has to know two things, where it is and where it's going. The latter of these is easily accomplished by transmitting to the vehicle a series of waypoints that define the path it is to follow. Once it has these two pieces of information, it's a fairly simple calculation to determine the range and bearing to the desired destination as shown in Figure 5.

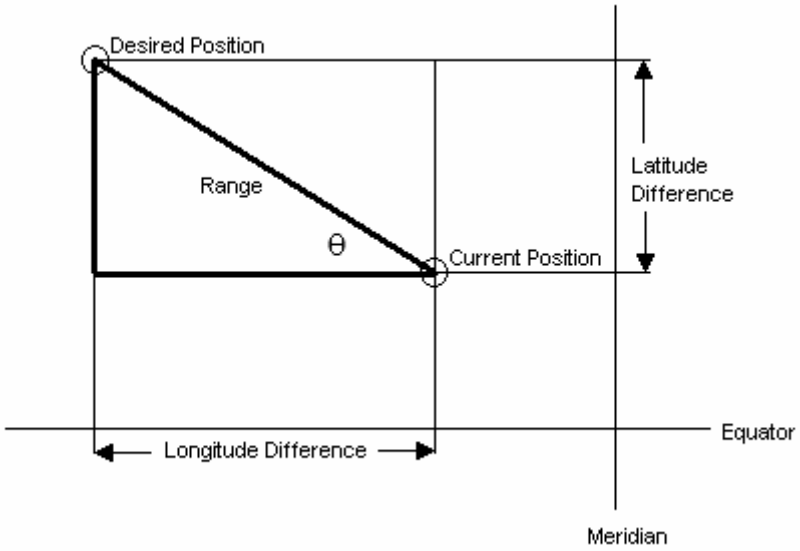

Figure 5. Range and Bearing Calculations

Range is determined by:

$$Range = \sqrt{latdiff^2 + longdiff^2} \quad (1)$$

The difference in latitude is easy to calculate since no matter where you go on the planet, lines of latitude are equidistant apart. Longitudinal lines, on the other hand converge as they approach the north and south poles making it a little more challenging. If we simplify and assume that the earth is perfectly spherical, then the range decreases as the cosine of the latitude. If we use the fact that one minute of latitude (and longitude when close to the equator) is equal to 2000 yards, we can arrive at a more specific formula for range:

$$R_{long} = 2000 * \cos(lat_c) \quad (2)$$

$$Range(yds) = \sqrt{(2000 * (lat_d - lat_c))^2 + (R_{long} * (long_d - long_c))^2} \quad (3)$$

In the above equations, R_{long} is the range in yards per minute of longitude, lat_c and $long_c$ are the current latitude and longitude coordinates in minutes of arc and lat_d and $long_d$ are the desired coordinates.

To determine desired heading, the vehicle must determine the angle theta and in which quadrant, relative to its current position, the desired position lies. Theta is easily determined by:

$$Theta = \tan^{-1}(latdiff \ / \ longdiff) \quad (4)$$

For desired positions that lie in the first or fourth quadrant, when the desired position is more easterly, the new heading can be determined by:

$$heading = 90 - Theta \quad (5)$$

Similarly, for those desired positions that lie in the second or third quadrant, meaning they are more westerly, heading is determined by:

$$heading = 270 - Theta \quad (6)$$

Obviously, if either the latitude difference or longitude difference are zero, the new heading is one of the four cardinal directions and easily determined by seeing if the nonzero difference value is positive or negative.

2. Heading and Speed Control

The two primary functions that need to be controlled on the autonomous platform are its speed and heading. Speed, for the most part, is very easy to control since the robot will only need to go forward at various speeds, or stop. For our purposes, speed can be controlled simply be ordering full speed when the robot is not very close to any desired waypoint, and slower than that when it gets closer to the desired waypoint. Heading control could be handled in a similar manner, but a more sophisticated system can be achieved through basic feedback control.

As seen in Figure 6, the desired heading enters the summing block and is compared to the actual heading which is determined by some sensor. The

difference is calculated and sent to a controller which in turn creates a signal to drive the plant and achieve the desired heading which in turn brings the difference or error signal to zero.

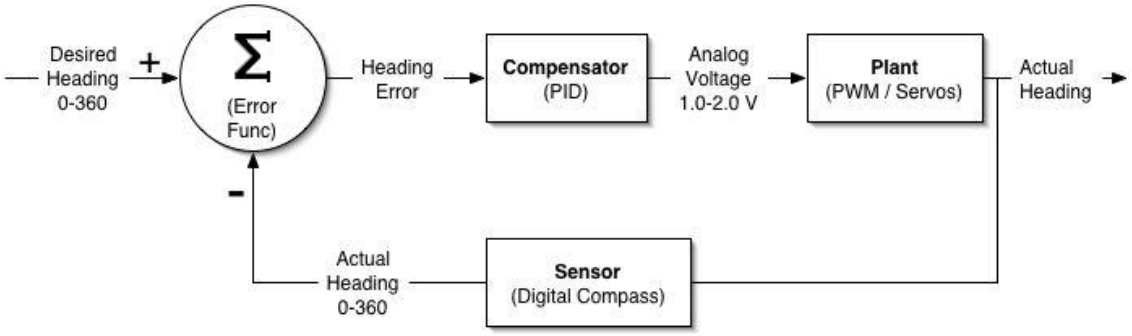

Figure 6. Feedback Control Loop

Inside the compensator block is a standard PID control algorithm that converts the heading error to an analog voltage between 1 and 2 volts that is sent to a pulse width modulator (PWM) and further on to the servos. The calculation of the voltage signal S is:

$$S = P_c e_{(t)} + I_c \int_{t_0}^{t} e_{(t)} dt + D_c \frac{de_{(t)}}{dt} \tag{7}$$

Where P_c, I_c and D_c are the proportional, integral and derivative coefficients and $e_{(t)}$ is the heading error. Since this is not truly a continuous process, the formula must be discretized to represent the iterative process that will take place in the code:

$$S = P_c e_i + I_c \sum_{i=0}^{n} e_i + D_c (e_i - e_{i-1}) \tag{8}$$

We see the the integral section is summed up over n iterations where n will most likely have to be determined experimentally for optimum performance. Additionally, the code will have to be designed to either store an array of n values for the previous errors or simply reset after n iterations and build up again.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ROBOT DESIGN ARCHITECTURE

A. CONCEPT OF OPERATIONS

In order to accomplish its required missions, the Whegs™ platform would be launched just outside the surf zone by some clandestine means, most likely by special forces, or by remotely controlled high speed surface boats. Just prior to launch, the platform would initialize and get a GPS fix of its location. Once a fix has been achieved, the robot would be launched into the water where it would sink or swim to the bottom. From the time it goes below the surface of the water and out of contact with GPS, it would use dead reckoning to navigate to its assigned set of waypoints. Once it hits the beach, it would reacquire GPS and continue its assigned mission. Future work would provide the ability for groups of autonomous vehicles to operate cooperatively to handle multiple tasks and ensure mission accomplishment through redundancy as well as allow mission modification through the use of unmanned aerial vehicles (UAVs) to pass mission updates.

B. MECHANICAL

The drive components of the platform were picked initially by means of convenience in order to emulate the drive system of the previous generations of Whegs™ [Ref 4][Ref 5] with the hope of reducing the actual driving components to three: one drive motor and two servos for steering. As seen in Figure 7, the motor is housed in the forward compartment (larger box on right of picture) and mounted on a base that allows it to slide fore and aft and is tensioned with an allen type set screw. Additional components are also labeled in this picture and mounting for the GPS can be seen in Figure 8, and will be discussed later. The driving force is transmitted to each of the three axles through the use of molded nylon chain and gears. The force on each axle is then transmitted through the compliance devices to each individual wheel-leg.

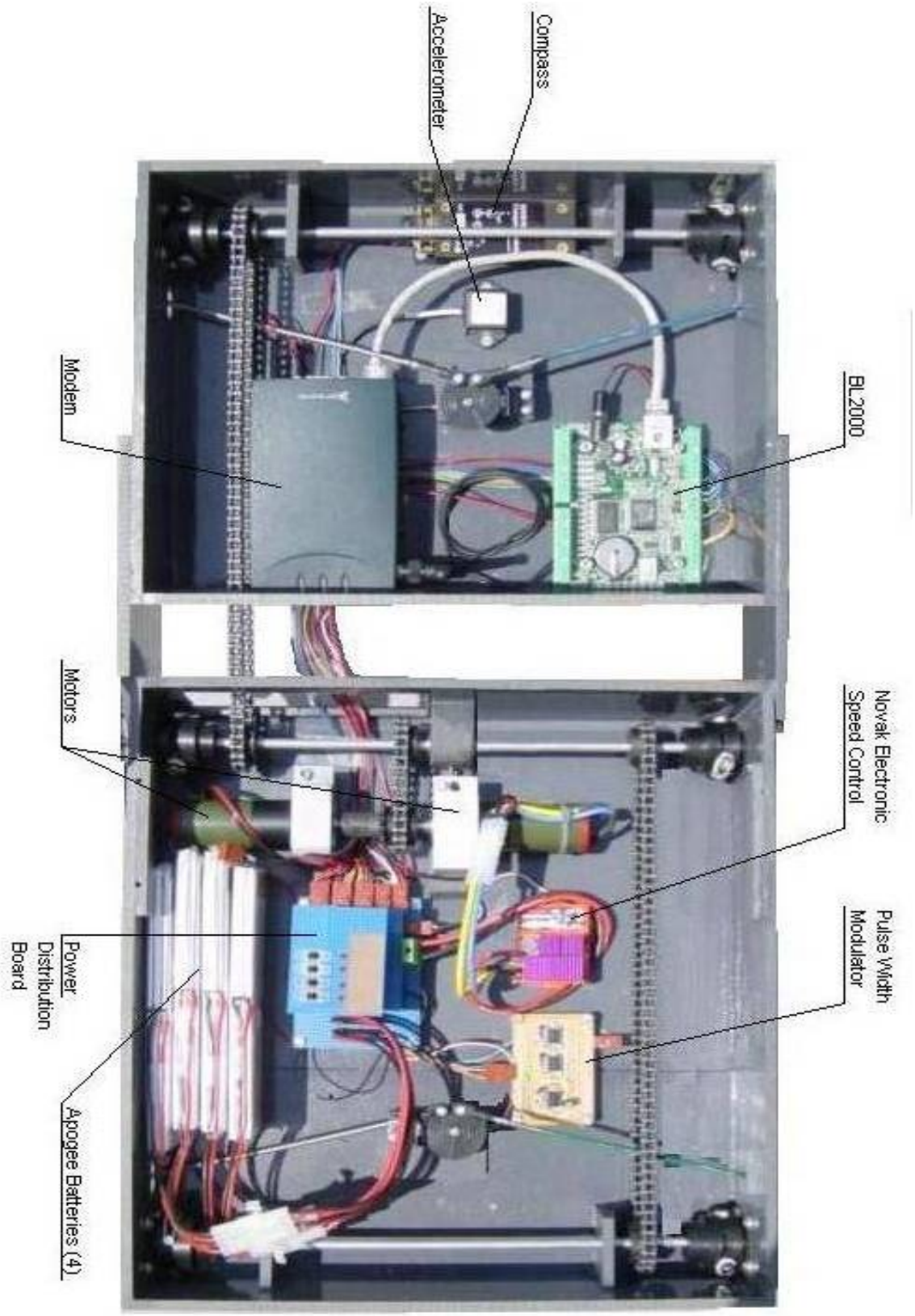

Figure 7. Physical Layout (without GPS bracket)

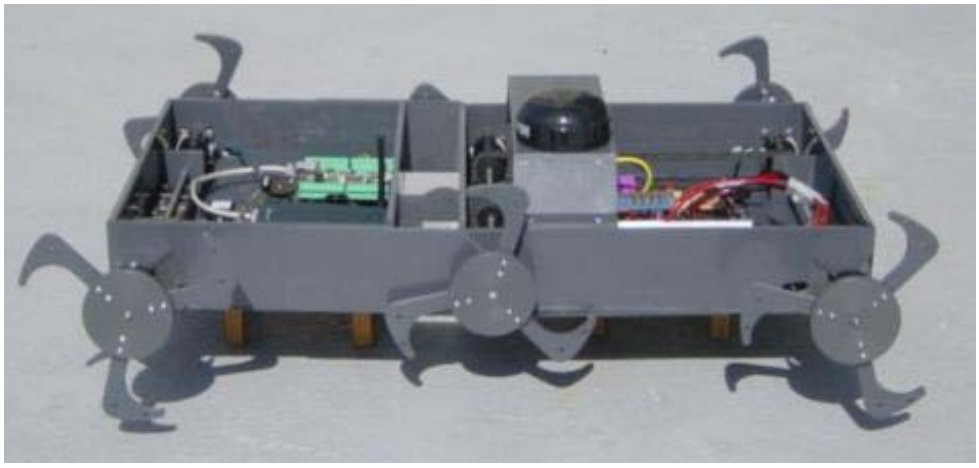

Figure 8. Side View showing GPS Bracket

1. Body Construction

The design of the robot body for this thesis was made to emulate, as close as possible, the work done by Alexander Boxerbaum in his mechanical design of the Whegs™ IV prototype [Ref 5]. In order to avoid confusion through the remainder of this thesis, the robot built here will be referred to as Lopez. Once both the Whegs™ IV prototype and the control system from Lopez are completed individually, they will be married into a complete prototype (this will be discussed more in Chapter IV). The current design includes a fully enclosed chassis that is waterproof up to 40 feet and dirt proof and will look similar to the artist rendition shown below in Figure 9.

Figure 9. Artist conception of Whegs™ IV [From Ref[5]

The first part of construction that took place for this thesis was in the design and manufacture of the compliance devices, followed by the wheel design, and finally the body design.

a. Compliance Devices

The compliance devices were designed similar to the devices from Boxerbaum [Ref 5] as shown in Figure 10.

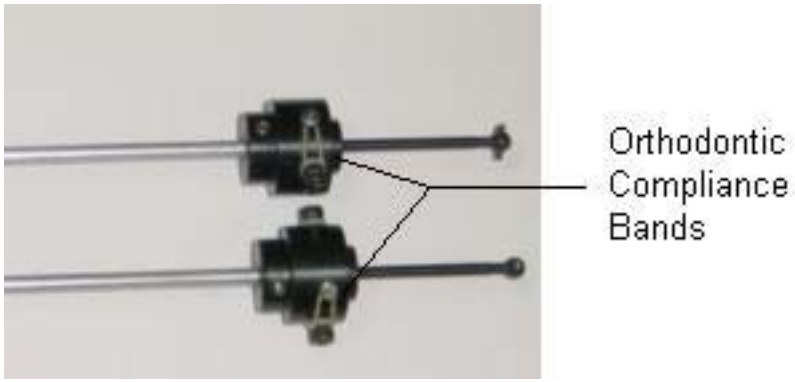

Figure 10. Compliance Devices

They were milled out of Delrin in two pieces with the inner piece firmly attached to the dogleg axle which drives the wheel-leg through the use of allen set screws, and the outer shell coupling to the driven axle. Slots were cut in the outer shell through which a M4 allen head bolt can pass where it attaches to the inner piece of the coupling device and allowing the two pieces to rotate coaxially through 60 degrees. Additionally the outer shell has two screws that are in line with the slotted holes. Once assembled, orthodontic rubber bands are stretched between the fixed screws in the outer shell, and the screws that pass into the inner piece in such a way that resistance is encountered as the inner piece rotates from 0 degrees to 60 degrees. The choice of Delrin as the material was based on the ease of milling, abrasion resistance and self lubricating properties of the material.

b. Wheel-legs

The wheel-legs were designed fairly independently of the work done by Boxerbaum [Ref 5] except for the basic three-legged concept. The driving concept for the design was to keep the wheel-leg, seen below in Figure 11, as modular as possible to allow for changes in radius of the wheel-leg as well

as changes to the size of the footpad which would be attached to the outer radius of each foot in order to provide sufficient floatation in loose sand and variable traction surfaces if needed.

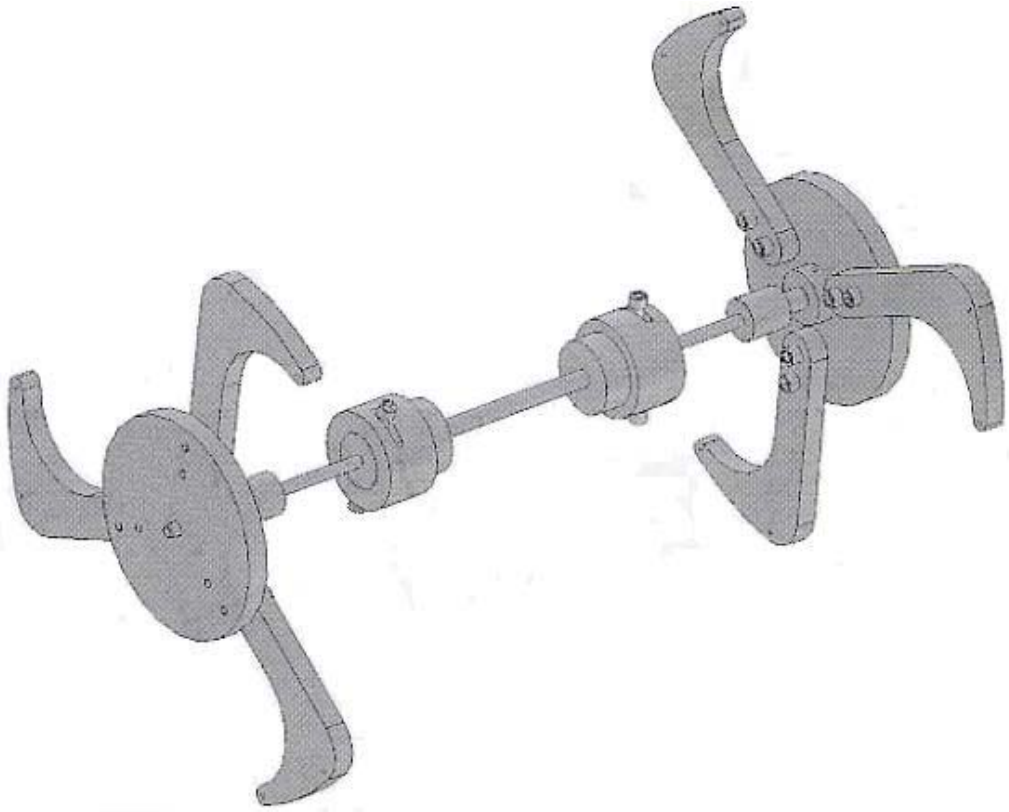

Figure 11. Axle with whegs and compliance devices

Both the hubs and the individual legs were constructed of polyvinyl chloride (PVC) and bolted together using M4 metric allen bolts. The hub is milled out in the center to accommodate a driving shaft. For the middle wheel-legs, the shaft is a simple quarter inch aluminum axle that passes through the body hinge and attaches to the inner piece of the compliance device. The front and rear wheel-legs can both be steered and so must be attached in such a way as to accommodate this. To do this, steering arms and a constant velocity shaft were modified from a standard remote control truck.

c. Body

The body was constructed of the same grade of PVC as used in the wheel-leg in the form of ¼ inch thick sheets. The basic design started with a 35cm by 70cm by 4cm tall rectangle which was separated into two sections

hinged directly in the middle. The separation forced the forward section to be larger than the aft to allow the two sections to pivot without obstruction. On the forward and aft most edges are struts that extend the steering mechanisms away from the body to provide sufficient room for the wheel-legs to turn without impacting the body. Previous designs of the Whegs^{TM} platform used actuated hinges to assist the platform in climbing obstacles and to prevent high centering [Ref 4], but this body design is intended to have a passively controlled hinge with either no resistance or limited resistance through the use of some form of progressive elastomer shocks. Currently the shocks are not implemented

2. Drive Train

a. Motors

The motor currently in use is a Maxon. S series motor with attached planetary reduction gear with specifications listed in Table 1. The choice of this motor was driven by availability more than a calculation of the torque requirements since mechanical design was not a primary focus of this thesis, however, they may prove sufficient provided the prototype body is light enough once completed.

Power (W)	11
Nominal Voltage (V)	12
Motor Stall Torque (mNm)	66.6
Max Cont. Torque (mNm)	17.53
Starting Current (A)	4.86
Reduction	66.22:1
Motor Weight (g)	174
Gear Weight (g)	194

Table 1. Motor/Gear Specifications

To determine the actual torque requirements, some additional facts need to be provided and some assumptions made:

- Vehicle Weight(W) 20lb(9.07kg)
- Wheel radius (r) 10.0cm
- Max Ground Speed (S) 0.5 m/s

Additionally, it is assumed that a single axle should be able to lift the entire body. Although this is not the case, the front axle will have to support a majority of the body weight when overcoming an obstacle, and it will provide enough power for the robot to handle more difficult terrain. To calculate torque, we can use the equation of motion for the drive system:

$$\mathrm{T}_m = (J_{eq}\ddot{\theta} + F_{eq}\dot{\theta} + \mathrm{T}_{ext}) / 66.22 \tag{9}$$

Where T_m is the torque felt by the motor, J_{eq} is the equivalent inertia, F_{eq} is the equivalent viscous friction and T_{ext} is any external torque (most likely from climbing inclines or obstacles). The division by 66.22 (from Table 1) converts the torque felt at the output end of the attached gearbox to the torque felt at the output of the electric motor section. The J and F terms are broken down into components as:

$$J_{eq} = \frac{1}{N^2} J_l + J_m \tag{10}$$

$$F_{eq} = \frac{1}{N^2} F_l + F_m \tag{11}$$

N is the gear ratio (here it is 1), J_l is the inertia of the load, J_m is the inertia of the motor, F_l is the friction of the load (meaning the friction losses inherent to the drivetrain) and F_m is the friction of the motor. Since the motor is a high efficiency motor, we will assume J_m and F_m are negligible. We can also assume that we want velocity to be constant, which eliminates the rotational acceleration terms and the formula reduces to:

$$\mathrm{T}_m = (F_l\dot{\theta} + \mathrm{T}_{ext}) / 66.22 \tag{12}$$

The frictional loss of the system is difficult for us to measure and not within the true scope of this thesis as the mechanical design will have a more efficient drivetrain. As such, we will call it T_{sys} for systemic torque and leave it essentially as a variable for now. The external torque can be calculated if we assume an incline of 45 degrees:

$$T_{ext} = \mu_k mgr \sin\phi \quad (13)$$

Using the values provided earlier for mass, wheel radius, and incline angle Φ, and assuming a no slip condition (which will provide the maximum torque) the value of torque turns out to be approximately 6.2 Nm. Combining equations 12 and 13 provides:

$$T_m = (T_{sys} + mgr \sin\phi) / 66.22 \quad (14)$$

This value (neglecting the systemic losses) equals 94.2 mNm which is drastically more than a single motor can accommodate. The improved mechanical design, however, will have a lighter body, reduced viscous friction losses and a more powerful motor.

3. Drive Control

a. Pulse Width Modulator

The PWM circuit on Lopez's platform is identical in design to the PWM used in the SMART design. It is constructed through the use of three 555 timer chips as shown in Figure 12. One chip provides a base pulse repetition frequency. This, in addition to an analog output signal from the BL2000 microprocessor is fed into each of the two remaining 555 chips in order to produce two PWM signals which are fed into the servos and to the Electronic Speed Control (ESC) for controlling the motors.

The pulse repetition frequency (PRF) is determined by the values of Ra, Rb and C. Here, the values are 10kΩ, 1kΩ and 1μF respectively. The amount of time to charge and discharge this portion of the circuit is determined by the formula:

$$T_{PRF} = .693(Ra + 2Rb)C \quad (15)$$

The pulse repetition frequency is easily determined from this by:

$$PRF = 1/T \quad (16)$$

Given the values listed above, the T_{PRF} is around 8.3 milliseconds, giving a PRF of 120Hz. The actual pulse width modulated signal times are

determined by the RC circuits in the right side of the schematic where R = 100kΩ and C = .047μF and are calculated by:

$$T_{PWM} = -\ln(1-V_{IN}/V_{CC}) \quad (17)$$

Where V_{IN} is the analog control voltage from the BL2000 and V_{CC} is the 5V provided by the ESC. T_{PWM} ranges from 1.0 to 2.0 milliseconds and turns the servo from maximum rotation counterclockwise to maximum rotation clockwise.

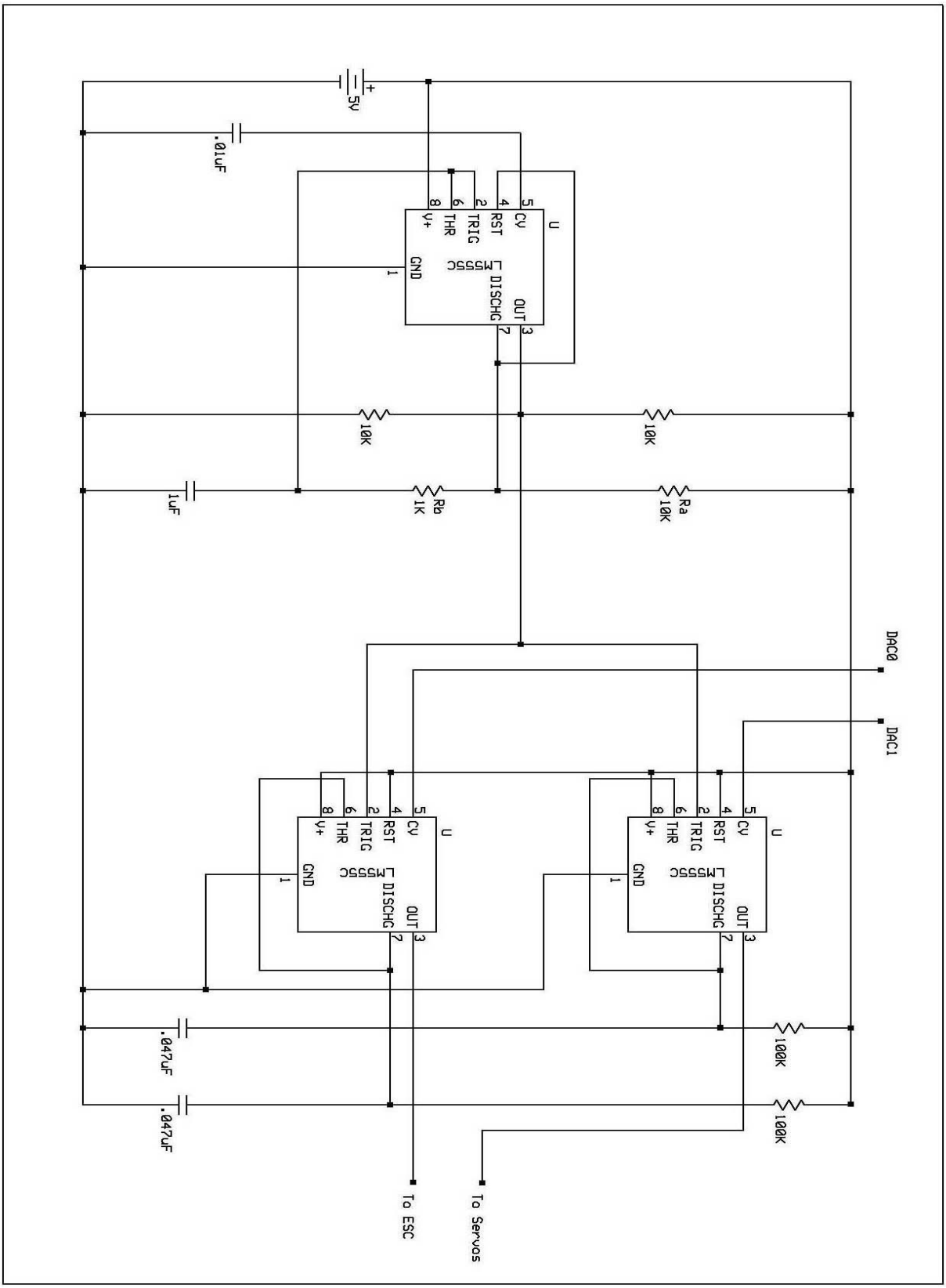

Figure 12. Pulse Width Modulator

b. Electronic Speed Control

The Novak Super Rooster, shown in Figure 13, is a COTS Electronic Speed Control (ESC) used for competition radio control car racing. The ESC is capable of handling a twelve- volt power supply and can safely source 320 amps in the forward and 160 amps in the reverse direction while providing an impedance of only a few milliohms. With one touch programming, the ESC is quickly calibrated to the input signals provided and the calibration data is placed in permanent flash memory until the user reprograms the ESC. The ESC interprets the signal supplied to it from the PWM circuit and converts the signal to the appropriate voltage to be applied to the motors [Ref 6]. Additionally, since the ESC is designed to drive servos, it is able to provide the five volt regulated power to supply the servos. Minor modifications were made to the ESC including removal of the on/off switch as well as the antenna since the power would be controlled through the main set of switches and all control signals would be coming directly from the PWM circuit.

Figure 13. Novak Super Rooster Electronic Speed Control [After Ref[7]

4. Communications Design

To handle communication between the user and the robot, a communication system was chosen based on the 802.11 configuration using one modem for the user and one onboard the robot. Transmission between the two uses a UDP format to prevent any hang-ups which might result from errors in the TCP handshaking routine. On startup, both the robot and the user computer

establish a series of sockets and ports through which data is sent. Ports 4001-through 4005 are established for the use of manual control, waypoint, gps, compass and error data, respectively. As seen in Figure 14, manual control order and waypoint information are sent from the user to the robot, and status information (position, pitch roll, etc.) are passed back to the user interface.

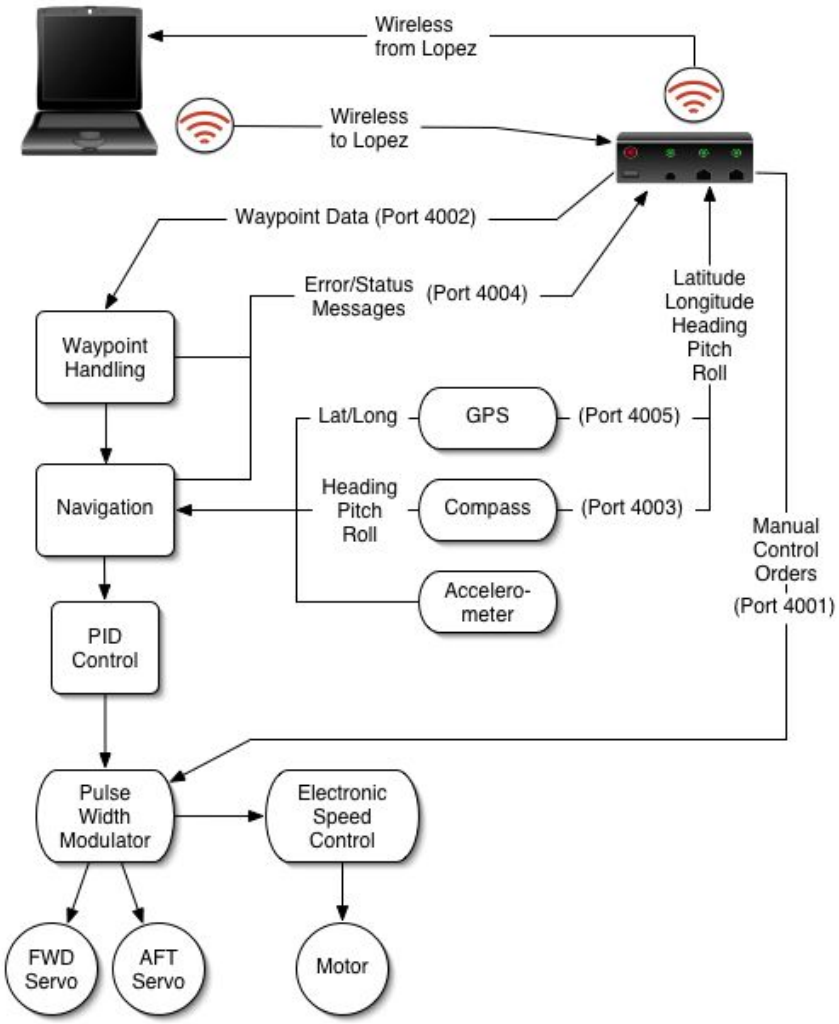

Figure 14. Communication Layout

C. ELECTRICAL

1. Batteries

In order to provide the best power to weight ratio, Lithium-Polymer batteries were chosen. The particular brand that was chosen was the Apogee battery from PFM Distribution, seen below in Figure 15.

Figure 15. Apogee Lithium-Polymer battery [After Ref.[8]

Four of the 3S2P 4160 batteries were used to provide sufficient on station time (approximately three hours depending on motor current draw). Table 2 shows the specifications of each battery.

Battery Capacity (mAh)	Pack Type	Volts (V)	Continuous Discharge (A)	Max Discharge (A)	Size LxWxH (mm)	Weight (g)
4160	3S2P	11.1	44	58	175x53x12.9	710

Table 2. Apogee Battery Specifications

2. Power Distribution

A few attempts were made in creating a useable power distribution bus. Primary design concerns were to keep the drive circuits and the control logic devices separated as much as possible. The first bus involved connecting two sets of Apogee batteries in series so there would be two 20-24V busses, one for control and one for drive, both of which would be regulated down to the required voltages as listed in Table 3.

Component	Voltage Requirement (V)	Current Draw (mA)
BL2000	9-40	TBD
Garmin GPS	6-40	70
Compass	6-15	>35
PWM	5	TBD
Accelerometer	8-30	15
Motor	up to 12	66-4860
Modem	6-15	TBD
Servos	5	TBD

Table 3. Voltage Requirements

By tying the batteries in series, the bus could provide the maximum possible voltage that the motor could handle. Most of the components had internal regulators, so only a few voltages were needed. The current layout of the power bus tied two sets of batteries in parallel to provide nominally 11 volts to all components which were self regulating except for the PWM and the servos. The power to the servos, as stated earlier, is provided by a voltage regulator onboard the ESC.

IV. LOPEZ CONTROL ELEMENTS

A. HARDWARE

1. BL2000 Microcontroller

The BL2000, as seen in Figure 16, is an advanced single-board computer that incorporates the powerful 22.1 MHz Rabbit 2000 microprocessor, 256K flash memory, 128K static RAM, 28 digital I/O ports, nine 12-bit A/D converter inputs, two 12-bit D/A converter outputs, a single pole, double throw (SPDT) relay output, and a 10Base-T Ethernet port. It is highly adaptable and easily programmed using Dynamic C.

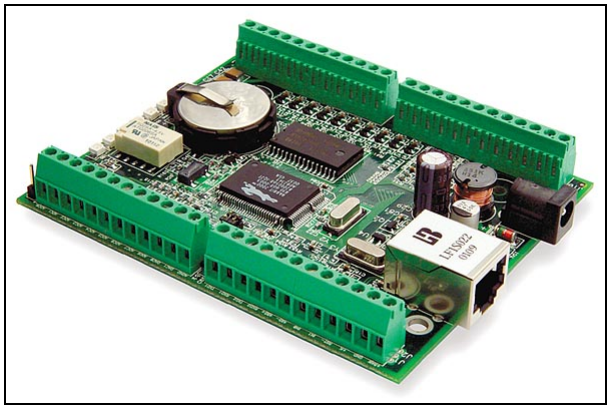

Figure 16. Z-World BL2000 Microprocessor [From Ref[9]

2. Garmin GPS16-HVS Global Position System

The Garmin GPS16-HVS, seen in Figure 17, is a complete GPS sensor including embedded receiver and antenna and is designed for a wide base of aftermarket system applications but targeted to automotive and marine applications.

Figure 17. Garmin GPS16-HVS Receiver [From Ref[10]

It is capable of tracking up to 12 satellites at a time while providing fast time-to-first-fix, one-second navigation updates and low power consumption. Additionally, the sensor is capable of FAA Wide Area Augmentation System (WAAS) differential GPS fixes yielding 3-5 meter accuracy. Power input

requirements range from 6.0 to 40 Vdc and output is RS-232 compatible, with selectable baud rate and ASCII output sentences based on the National Marine Electronics Association's (NMEA) 0183 ASCII interface specification. The primary sentence used is the GPGGA sentence which provides Global Positioning System Fix Data in the following format:

$GPGGA,1,2,3,4,5,6,7,8,9,M,10,M,11,12*hh<CR><LF>
1 = UTC time of position fix, hhmmss format
2 = Latitude, ddmm.mmmm format
3 = Latitude hemisphere, N or S
4 = Longitude, dddmm.mmmm format
5 = Longitude hemisphere, E or W
6 = GPS quality indication (0=fix not available,
1=Non-differential GPS fix available,
2=Differential GPS (DGPS) fix available,
6 = Estimated)
7 = Number of satellites in use
8 = Horizontal dilution of precision (0.5 to 99.9)
9 = Antenna height above/below mean sea level
10 = Geoidal height
11 = DGPS data age (in seconds)
12 = DGPS reference station ID
*hh is a parity checksum and is not required

Sentence rate is programmable from 1 to 900 seconds in one-second increments, and for maximum accuracy, is set to the default 1 second rate for this application.

3. Honeywell HMR3000 Digital Compass

The Honeywell HMR3000, shown in Figure 18, is a small single card electronic compass unit capable of providing heading information as well as pitch and roll information up to 40 degrees with an accuracy of 0.5° and 0.1° resolution. The sensor is magnetoresistive, utilizing the anisotropic magneto resistance of the ferrous material of the sensor. The sensor material is placed in a basic Wheatstone bridge configuration from which the heading is calculated. The card utilizes a 6 – 15Vdc unregulated input and provides a standard serial RS-232 output to the user. The output is programmable to provide various data including pitch, roll, magnetic heading, deviation, variation, true heading, or any combination of these required in various formats. The output signal can be

updated from six to three hundred times per minute and, for this application, updates once per second.

The standard programmed output string is NMEA 0183 compatible and is of the following format:

$PTNPHTR,HHH.H,N,P.P,N,R.R,N*hh<CR><LF>.
HHH.H = Heading information in degrees
P.P = Pitch information in degrees
R.R = roll information in degrees
N = Heading, Pitch, and Roll status
*hh is for checksum parity
<CR> = Carriage Return
<LF> = Line Feed

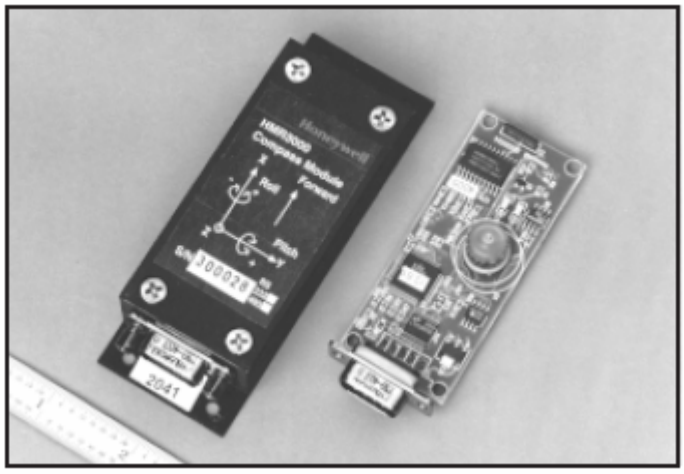

Figure 18. HMR3000 Digital Compass [From Ref[11]

4. Crossbow MEMS Accelerometer

The Crossbow Industries LP Series of accelerometer, shown in Figure 19, is a general purpose, linear acceleration and/or vibration sensor. The sensing element is a silicon micro-machined capacitive beam which is conditioned to provide a direct analog voltage output that requires no additional signal conditioning. The model used is the CXL10LP3-R which is capable of withstanding forces up to ±10g, measuring acceleration in 3 orthogonal axes and operating with a supply voltage of +8.0 to 30VDC. The output voltage levels are typically 2.5V for zero g with a ±2.0V output span. Sensitivity is 200 ±10 mV/g.

Figure 19. Crossbow MEMS Accelerometer [From Ref[12]

For the purposes of turning this device into a rudimentary dead reckoning device, each of the three outputs are fed into the BL2000 which performs a double integration to convert the accelerations into distances in each of the three axes. Corrections are applied from the pitch and roll values received from the compass in order to rotate the distances covered to a frame of reference normal to the mean surface of the earth. Measurements of change in altitude are essentially neglected from this point. Next, corrections are applied for the heading which can in turn be applied to a previous latitude and longitude to update the platforms position until a good GPS fix is achieved.

5. Modem

Lopez uses two RangeLAN2 7921 Ethernet adapters, shown in Figure 20, to communicate between the user and the robot. The RangeLAN2 is a long range, high performance modem that uses frequency hopping spread spectrum technology (FHSS) in the 2.4GHz range to provide clear, relatively secure communications up to 300 feet in optimal conditions. The two modems are set up in an ad-hoc configuration with the modem attached to the Java interface serving as the master and the one attached to the robot serving as the slave. The only difference is that the master coordinates the frequency hopping of both modems.

Figure 20. Proxim RangeLAN2 Modem [From Ref[13]

B. SOFTWARE

The control algorithm for Lopez is similar to that of Bender© in that it uses two separate programs. One is the imbedded Dynamic C program that runs the operation on the BL2000. Once given a route to follow, it is capable of controlling all aspects of Lopez until it reaches one of several possible conditions that will return it to manual control mode of operation. The other program is the JAVA interface which allows the user to monitor, manually control or program a waypoint path for Lopez to follow. Communication between the two is handled using standard TCP/IP sockets over a 802.11b wireless configuration.

1. Imbedded program

The imbedded program is a Dynamic C program that initializes all the onboard equipment, establishes the communication sockets and then enters a continuous cooperative multitasking environment that controls the communications, navigation and movement of Lopez through the use of costatements. Costatements are a design feature of the Dynamic C programming language which allows the program to cooperatively multitask by voluntarily suspending execution of the costatement until certain parameters within the statement are met. Figure 21 shows the basic flow of the program and the major variables that are used. Once initialized and in the main loop of the program, the program will cycle through the costatements, all of which have one or two lines at the beginning to determine if it is necessary to enter that particular costatement. If it is not needed, the program exits that section and checks the other costatements. One item that is not shown in the flow of the program, but is a vital

part of protecting Lopez is the bus voltage checking that occurs at the beginning of the main program loop. This is needed to reduce the possibility of damage to the lithium polymer batteries and will either warn the user that the voltage is getting low, or when critical, it will stop Lopez. Although this will help prevent damage, it is not capable of disconnecting the batteries or completely shutting down.

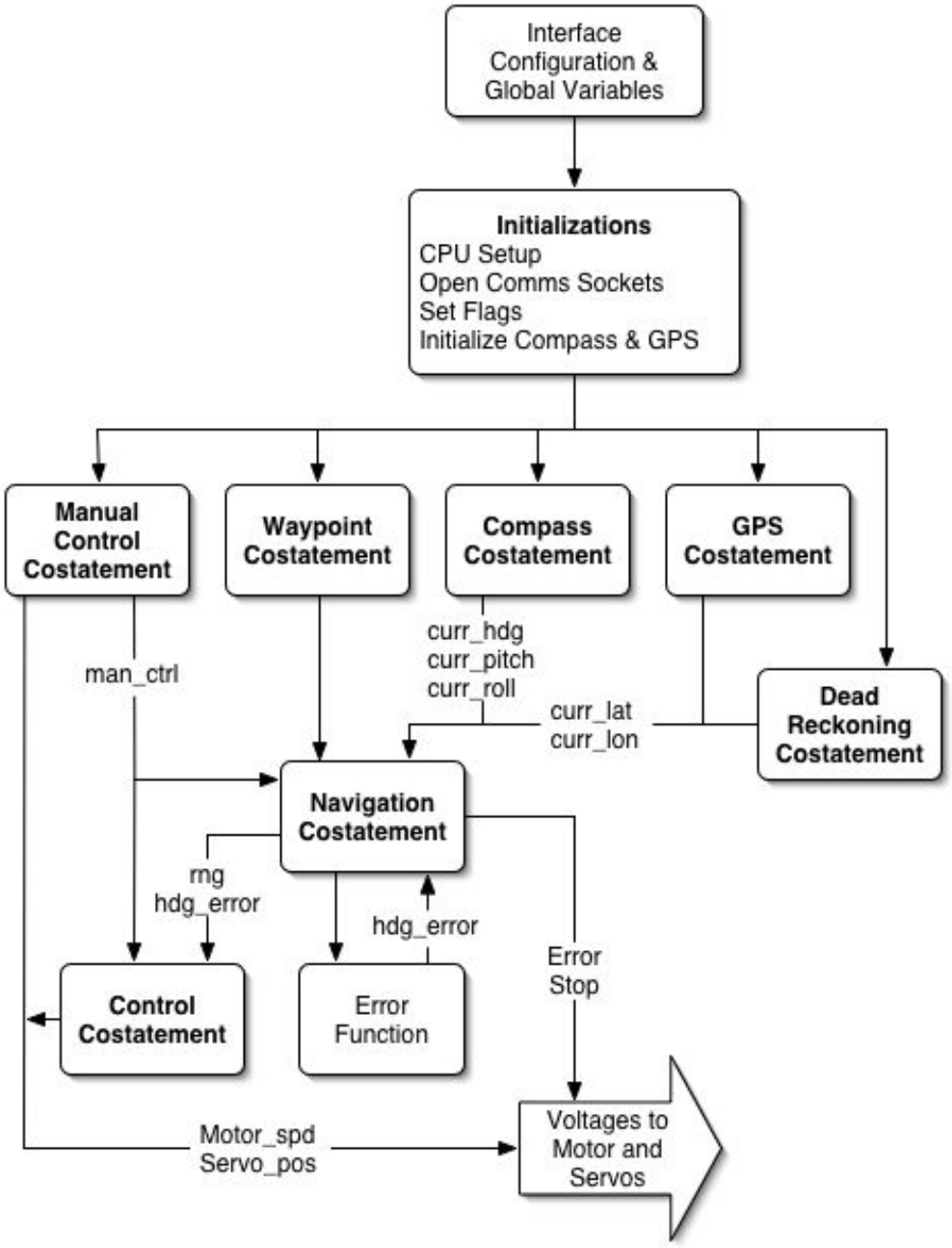

Figure 21. Dynamic C Program Flow

a. Manual Control Costatement

The manual control costatement is only triggered when a manual control order is received from the user on port 4001. It extracts the information about the required speed and servo position, transmits that directly to the pulse width modulator, sets the man_ctrl flag (which prevents entering the navigation or control costatements), and transmits a message back to the user that the order has been received.

b. Waypoint Costatement

The waypoint costatement is also triggered only by data coming in from the user on port 4002. Its major function is to take the waypoint information that is coming in from the interface and store it into the waypoint structure so it can be accessed by the navigation costatement. Once the data is stored, it resets the manual control flag and notifies the user that the waypoints were received and that the robot is proceeding in auto-navigation mode.

c. Compass Costatement

The compass costatement is triggered on a time dependent basis which is currently set at 10 milliseconds. Once active, it reads the ASCII data that is coming in on serial port B of the BL200 (which uses RS-232 format) and puts the information into an array. While it is reading the data, it is also checking a timer to ensure that the program does not hang up if there is a problem with reading the compass. Once the information is retrieved from the compass, it is transmitted as a whole back to the user via port 4004 for display. Next, the array is passed to the compass_get_hdg function which checks for errors in reading and, if the sentence doesn't have any errors, it extracts the robots current heading, pitch, and roll information.

d. GPS Costatement

The GPS costatement is also triggered on a time dependent basis, but it is set at half a second. It is nearly identical to the compass costatement in how it operates. It reads serial ASCII data in on port C, has a safety timer and sends the data back to the interface via port 4003. It calls gps_get_position, which in turn uses the function gps_parse_coordinate, to fill a structure that

contains all the latitude and longitude information. If there were no errors in parsing the information, the costatement breaks the structure into the variables for current latitude and longitude. If there are errors, the costatement increments the error-count variable and exits. If this error-count variable is greater than six, the dead reckoning costatement is triggered.

e. Dead Reckoning Costatement

When triggered, the dead reckoning costatement looks at the DR_avail flag which is set the first time a GPS fix is achieved, and reports to the user that it either stopped and is in manual control (when no fix is available) or that there was a loss of GPS and that it is now in DR mode. The next task the dead reckoning costatement accomplishes is to convert the voltage from the MEMS accelerometer to a numeric value for acceleration measured in yards/second^2:

$$A_x = (Xchan - A_{xc}) * (10.72 / .2) \tag{17}$$

Where A_x is the voltage in the x direction, A_{xc} is the calibration value (nominally for the Crossbow accelerometer, 0g = 2.5 volts), and Xchan is the value of the voltage from the X direction. The value 10.72 is the acceleration of gravity in yards/second^2 and the .2 is the sensitivity of the accelerometer (200mV per g). This calculation is done in each of the three directions.

From there, it performs a double integration. Looking at the program, it can be seen that velocity values are carried from one call to the next, but distance values are not. Once the double integration is complete, the values of distance in the x, y and z direction have to be rotated to align with the gravitational field so the distances are normal to the average plane of the earths surface seen below in Figure 22.

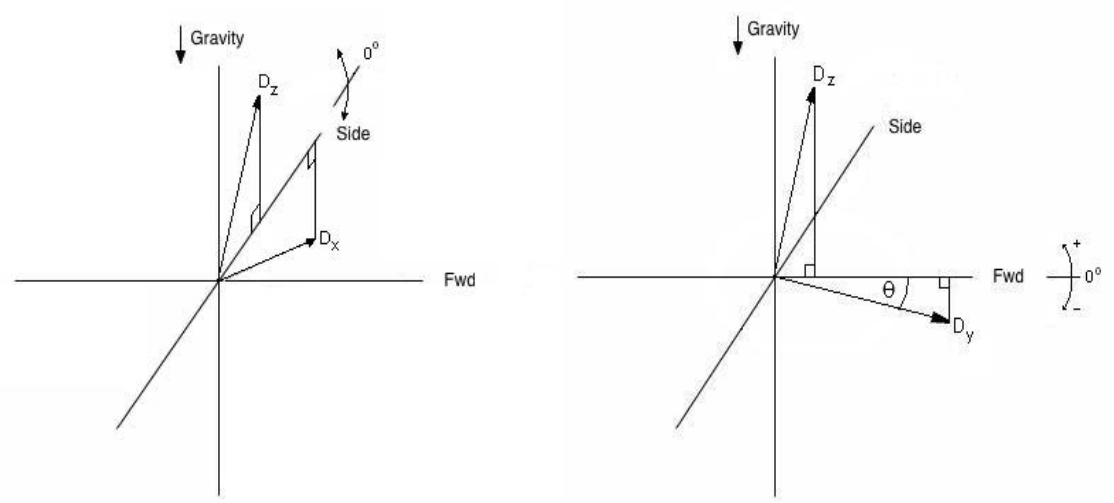

Figure 22. Rotation of Axis

The rotation is done in two parts, taking the distances from the accelerometer (D_x, D_y, and D_z), and convert them to distance traveled forward and distance traveled sideways. When traveling on land, the distance traveled sideways should be minimal, but when underwater, there will be movement due to currents and fluttering as is sinks or swims to the bottom. The angular values are retrieved from the compass where pitch is denoted as theta and roll is denoted as phi. The calculations are:

$$D_{fwd} = (D_y \cos\theta) - (D_z \sin\theta) \tag{18}$$

$$D_{side} = (D_x \cos\phi) - (D_z \sin\phi) \tag{19}$$

Once these are complete, the next rotation is to align the distance covered forward and sideway with distance covered in latitude and longitude as in Figure 23.

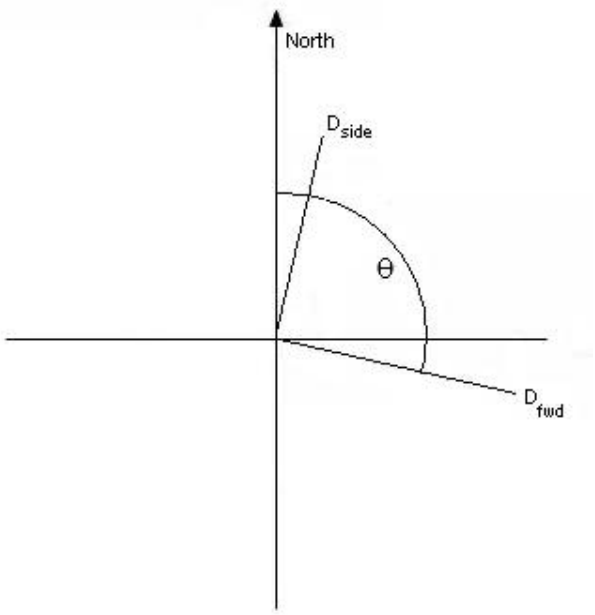

Figure 23. Rotation to Latitude and Longitude

In this figure, theta now represents the heading angle in degrees magnetic and the distance covered in latitude and longitude is:

$$D_{lat} = D_{fwd} \cos\theta + D_{side} \sin\theta \tag{20}$$

$$D_{lon} = D_{fwd} \sin\theta - D_{side} \cos\theta \tag{21}$$

The final calculations convert D_{lat} and D_{lon}, which are measured in yards, to DR_lat_adj and DR_lon_adj, which are measured in decimal degrees and then sum these with the previous position. Before exiting, the position information is converted to fit into the same NMEA format that is produced by the GPS and it is sent out to the user via GPS port 4003.

f. Navigation Costatement

The navigation costatement is triggered when manual control is not being used. It takes the inputs from the GPS, compass, and DR costatements to determine where the robot needs to go when in auto-navigation mode. First, it determines range using the calculation from equation 3 and decides if the robot is close enough to the desired waypoint. If so, it stops, goes to the next ordered waypoint or restarts from the first waypoint depending on the input from the Java

interface. There is also an emergency stop function in case the program tries to go to a waypoint not contained in the array of available waypoints.

If the robot is not close enough to the desired waypoint, the costatement recalculates the desired heading using equations 4 – 6. The new heading is passed to the error function which converts it to a heading error that can be passed to the control costatement.

g. Control Costatement

The control costatement is triggered if the manual control flag is not set. It takes the heading error received from the navigation costatement and passes it to the PID function which determines the value of Servo_pos using the control loop discussed in section II-B-2.

2. JAVA Interface

The JAVA Interface, as seen in Appendix B, was created by Kubilay Uzan [Ref 14] to serve as the graphic user interface (GUI) for Bender©. Very little was done to modify this program to allow its use for Lopez. The only significant change was the modification of calculations for the use of the manual control buttons on the far right of the screen and manipulation of the pitch and roll parsing. As seen on the left of Figure 24, there are windows that monitor GPS data, compass data and communications from Lopez. Also in the middle of the left side, there are several buttons that allow manipulation of waypoint paths

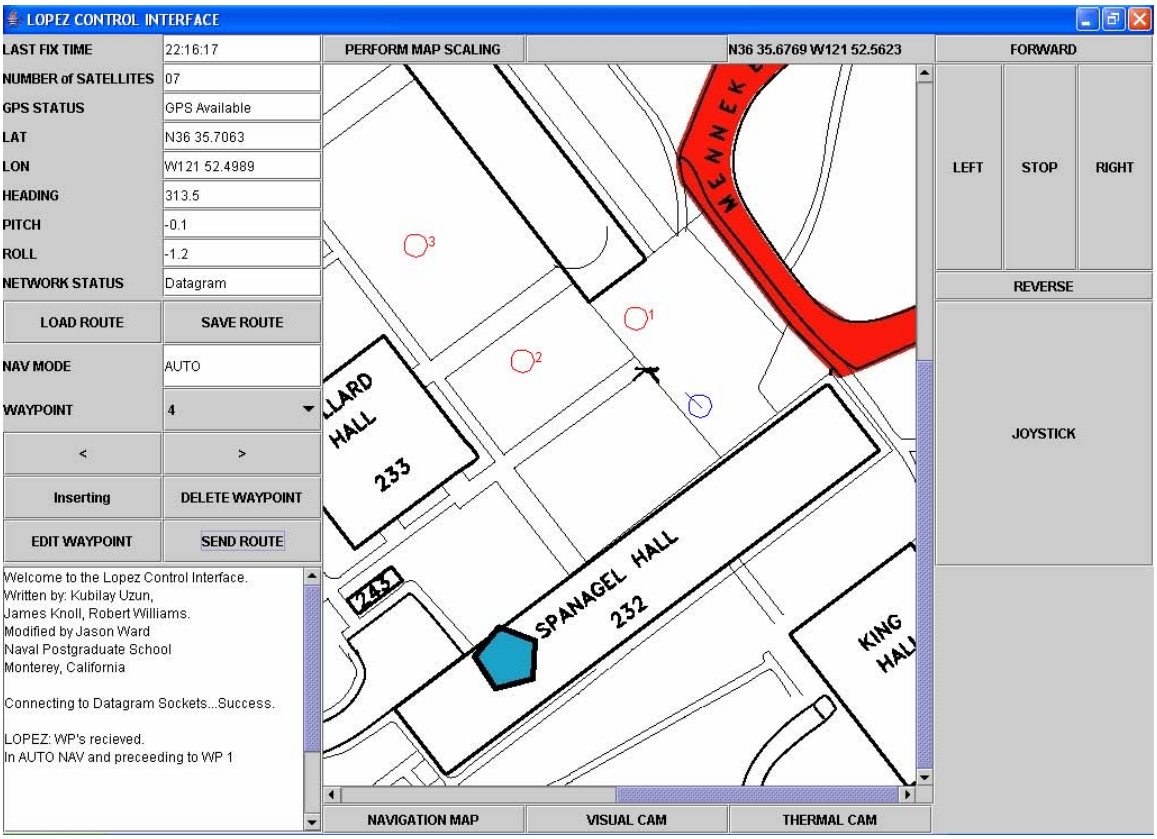

Figure 24. JAVA Interface

The majority of the screen is taken up by the map which normally shows the robots location and direction through a position indicator with heading leader. Waypoints also show up on the map so the user can see the route. The map is vector scaled so dragging the cursor over the map will cause the program to calculate the latitude and longitude of the location (seen at the top of the map). Double clicking on the map allows the user to enter waypoints without having to know the exact GPS coordinates.

V. RESULTS

A. DRIVE TRAIN ANALYSIS

The first attempts at having Lopez move under its own power were somewhat less than successful. There were two primary problems: insufficient power and chain skipping. Although this was not a major objective of this thesis, since there was already a more robust body design in construction, it did make it more challenging to test the operation of the control system. The first steps taken by the robot required a little pushing to get the platform moving in the first place, and it would stop after a few steps. Additionally, when the drive train would come under load, a small amount of slack could be seen on the chains between the middle and outer axles. Due to the quick design process, there was no way of preventing this slack. Given enough slack, the chain would jump the cogs of the driven axles causing them to go out of their proper phase.

B. PID IMPLEMENTATION

Previous versions of the control costatement used look-up tables to steer Bender© because attempts to use PID control caused the platform to become unstable when it was between 5 and 10 yards from its desired waypoint and it would back away from the waypoint and make several turns because of oversteering. This was partially due to the control system, but also in part from the time lag between calculations of the desired heading.

The PID control implemented on Lopez, after correcting a few minor programming errors, has shown to be smooth and stable. Currently, it is actually only serving as a proportional controller, since the integral and derivative coefficients are set to zero, but the ability is resident should future experimentation reveal the need for integral control. Due to the light weight of the platform, it is very doubtful that derivative control will ever be needed.

One possible problem that was not tested was the possibility of overshooting the waypoint. Two likely fixes would be to either slow the platform down as it approaches the waypoint (more than it is already) if accuracy is

required, or make the allowable range error variable larger in the navigation costatement.

C. POWER BUS OVERHEATING

The first design of the power distribution called for two 22 volt busses that were both regulated down to 12 and 5 volts. The reason for this was to maximize the voltage that could be applied to the motors (since the batteries only operated at 11.1V nominal). When operating the motors for any amount of time, however, the regulators would create a large amount of heat. Although, at the time, the heating effects were not a major concern, once the control system was moved to the new body, componenets would be much closer and the entire unit would be sealed, which could cause problems with component accuracy, not to mention the possibility of melting wires and causing fires. Because of this, the decision was made to use the same number of batteries, but in a parallel configuration so regulation from 22V to12V was not required.

D. SERVO/MOTOR INSTABILITY

Currently, the regulated 5 volts, which provides power to the servos and to the PWM circuit, is provided by the Novak Electronic Speed Controller. This seemed to be an easy way to provide 5 volts without having to build a separate regulator, thus saving space, but it turned out that the output voltage was subject to occasional fluctuations for reasons not understood, due to the proprietary nature of the Novak design. Although this is not a major concern as a power supply for the servos, it causes a major problem for the PWM. As seen in equation 17, any change in the supply voltage causes a significant change in the pulse width of the signal, which in turn would cause the servos and motor to twitch. Currently the problem still exists, but it is a rarely occurring problem, so the decision was made to ignore it until the control system is moved into the new body, at which point the power distribution bus will have to be redesigned anyway to accommodate the space requirements of the new platform. When this is built, it will have a regulator built in specifically for the PWM (and any future logic boards). The heating problem from above should not be a concern due to the small current draw that will needed for the logic boards.

E. DEAD RECKONING PERFORMANCE

The first test run of the dead reckoning system was a partial success. The platform turned towards the expected point and stopped in the vicinity of the final waypoint, but no measurement was done to test the accuracy. The problem with the first run was that position information was not reported back to the user. The interface program requires the NMEA formatted message to be reported back so that it can be parsed and the screen updated. At publishing time, the program was being modified so the dead reckoning costatement could take the calculated position and format it for transmission back to the user. Once accomplished, further testing will be completed to measure the accuracy of the dead reckoning over differeing distances.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. FUTURE WORK

A. CARBON FIBER BODY

After the completion of work by Alex Boxerbaum on the mechanical design of Whegs^{TM} IV [Ref 5], the control system covered here will be integrated with that body and a carbon fiber shell will be constructed to seal all the components from the environment.

B. BEACH FIELD TEST

Once the two designs are integrated, the robot will be run through series of tests to evaluate its performance in the surf zone environment starting with beach tests to ensure that it can operate in loose sand. As the operating capability is determined, more obstacles will be tested to measure its ability to passively overcome obstacles of varying size. Once this range of obstacles is determined, it will serve as a baseline for the requirements of active obstacle avoidance.

Next, the robots ability to maneuver in the water will be tested. Pool tests will be conducted first to measure buoyancy and stability underwater. It is expected that modifications will need to be made to reduce the likelihood of flipping or excessive fluttering while sinking or swimming to the bottom.

C. SONAR IMPLEMENTATION

The most likely option for obstacle detection will be some form of sonar. Research will need to be conducted to find a suitable transducer that is easy to integrate, matches the size, weight and power limitations of the platform and works as well in or out of the water. Once a choice of transducer is made, an obstacle avoidance routine will be invoked in the code, drawing heavily on the experience of ongoing work with the SMART program.

D. SATELLITE PROCESSING

In order to reduce processing time of the BL2000, especially as future sensors are added, some of the processing could be handled by microcontrollers specifically suited to the individual device that would provide the processed data

to the BL2000 when required. Hopefully, this will reduce the lag time in communications and response to orders.

APPENDIX A. IMBEDDED CODE

```
/*******************************************************\
LOPEZ_NAV_ver.c
    Lopez Nav is the navigation interface with the BL2000 processor
    used to control the second generation Physics Department robot
    known affectionately as "Lopez" from the popular online series, Red
    vs Blue. This program is a direct adaptation of the BENDER_NAV
    program.
*********************************************************
Version History: Last Saved 21 April

    --- Lopez Version 0.1.4 ---
    May 2005
        LT Jason Ward
        This version adds PID

    --- Lopez Version 0.1.3 ---
    May 2005
        LT Jason Ward
        This version adds DR, but no PID

    --- Lopez Version 0.1.2 ---
    April, 2005
        LT Jason Ward
        Rewriting code version 0.0.2 to accommodate Dynamic C 9.21
        This does not have DR or PID functions

    --- Lopez Version 0.0 ---
    January - June, 2005
        LT Jason Ward
        Changes:
        This version eliminated the Sonar costatement as Lopez does not yet
        have sonar capability.
        Control - complete redesign to accommodate a servo steering
        system vice a differential track drive system

    --- Bender Version 1.0 ---
    July - September, 2004
        CDR Jerry Stokes
        LT Sean Niles
        LT Irv Pollard
        LT Jason Ward
        LT Brett Williams

        Changes:
        This version implements the Sonar costatment, allowing Bender to react
to obstacles in his path.  Algorithms to handle impediments to Bender's planned
path have also been implemented.
        Nav - pared down to handle only calculations of GPS positions and
generating headings and ranges for following waypoint paths. GPS is now
initialized to only give 2 sentences (one cannot be turned off) so that data
culling when parsing the GPS sentence is reduced.
        WayPoint - algorithm was optimized and altered to allow the user to
input an exact path desired.  Previous versions used the closest waypoint as
the next waypoint to which to drive.
        Control - PID control was implemented as its own costatement, partly to
alleviate the delay associated with taking manual control when Bender was
performing calculations in Nav.
        Networking - Ports were shifted to 4001 and higher to avoid conflicts
with reserved ports. UDP is still implemented; however with greater throughput
```

```
realized from 802.11g wireless hub, it may be advantageous to switch to TCP/IP
in later versions. IP addresses were shifted to 192.168.0 domain for the
wireless router. Should future users decide to move Bender to the university
network, static IP addresses will need to be obtained for the camera, Bender,
and router.

    --- Bender Version 0.6.6 ---
    Created for SE4015, Summer 2003
        James Knoll
        Kubilay Uzun
        Robert Williams

        This program was written to run on the BL2000 and control the
    Nav, Sensors, and Motors of Bender.
        Compiles with two warning in Dynamic C 7.04P3.  Newer versions of
Dynamic C will require modification in the networking since UDP has changed.
*******************************************************************
BL2000 CONNECTIONS

    Nav   to   Drive
    dac0        motor
    dac1        servo

    Nav   to   GPS
    tx2         RED
    rx2         GRN
    grnd        BLK

    Nav   to   Compass
    tx1         GRN
    rx1         RED
    grnd        BLK

    Nav   to    Power Bus
    adc5        Drive Bus
    adc6        Ctrl Bus

    DR    to    Accelerometer
    adc0        X-axis
    adc1        Y-axis
    adc2        Z-axis

*******************************************************************/

#define READDELAY 15

#define MAX_SENTENCE 100


/*
 *  Networking Settings
 */
#define TCPCONFIG 0

#define USE_ETHERNET 0x01                  //Configuration Setup
#define IFCONFIG_ETH0 \
   IFS_IPADDR, aton("192.168.0.91"), \
   IFS_NETMASK, aton("255.255.255.0"), \
   IFS_ROUTER_SET, aton("192.168.0.1"), \
   IFS_UP
#define INTERFACE_ADDRESS  "192.168.0.90"         //IP of Laptop

#define WP_PORT 4002
```

```
#define MAN_PORT 4001
#define COMPASS_PORT 4004
#define GPS_PORT 4003
#define ERROR_PORT 4005

#define MAX_UDP_SOCKET_BUFFERS 5
#define DISABLE_DNS                //DNS not needed for direct control
#define DISABLE_TCP                //TCP not needed, all comms are UDP

#use "dcrtcp.lib"
#use "udp.lib"

#memmap xmem

/*
 *  Serial Port Settings
 */

#define BINBUFSIZE 127
#define BOUTBUFSIZE 127
#define CINBUFSIZE 127
#define COUTBUFSIZE 127

/*=====================================================================*\
   GPS Variables
\*=====================================================================*/

    double curr_lat;
    double curr_lon;

    const int xmit_delay = 100;

    char sentence[MAX_SENTENCE];
    char dir_string[2];

    typedef struct {
        int lat_degrees;
        int lon_degrees;
        double lat_minutes;
        double lon_minutes;
        char lat_direction;
        char lon_direction;
    } GPSPosition;

    GPSPosition current_pos;   // Declare new GPSPosition variable

    const int gps_delay = 0.5; // delay seconds between gps readings

    int gps_error, gps_error_count;
    int GPS_updated;

    const float pi = 3.14159;

    const char GPS_Reset[]="$PGRMI,,,,,,,R\r\n";  //Unit reset
    const char GPS_Sent_Clr[]="$PGRMO,,2\r\n";    //clears all output
                                                    sentences
    const char GPS_GGA_Enable[]="$PGRMO,GPGGA,1\r\n";//enables the GGA
                                                       sentence
//const char GPS_Parameter_Set[]=
                "$PGRMC,A,,,,,,,,A,4,,,,\r\n";//Sets baud rate to 9600
    unsigned long gps_wait_time;
    const int gps_timeout = 1;
```

```
/*====================================================================*\
   DR Variables
\*====================================================================*/

    int DR_nav;
    int DR_avail;       //Prevents entering DR without a good first fix

    float Vx, Vy, Vz;   //Velocity variables for dead reckoning
    double DR_lat_adj, DR_lon_adj;
    int lat_int, lon_int;
    double lat_mant, lon_mant;
    char lat_int_ascii[5], lon_int_ascii[6];
    char lat_mant_ascii[20], lon_mant_ascii[20];
    char lat_str[10], lon_str[11];

/*====================================================================*\
   Compass Variables
\*====================================================================*/

    float curr_hdg, curr_pitch, curr_roll;
    char compass_sentence[MAX_SENTENCE];
    int compass_error;

    const int compass_delay = 10; //mili-seconds to delay between
                                  //compass readings
    const char init_str[] =
            "#BAD=8*7A\r\n";// sends HPR sentence 5 times per second

    int string_pos;
    char input_char;

    unsigned long compass_wait_time;
    const int compass_timeout = 1;

    int Compass_update;

/*====================================================================*\
   Communications Variables
\*====================================================================*/

    longword host;
    // These transmit in the blind
    static udp_Socket compass_data, gps_data, error_data;
    // These receive data
    static udp_Socket wp_data, man_data;
    char cmdBuf[1024];
    char cmdstr[20];
    char wptBuf[4096];
    char wptstr[500], *wptptr;
    char error_buf[200];

/*====================================================================*\
   Navigation Variables
\*====================================================================*/

    const float brg_error = 5.0;    //Allowable Bearing Error
    const float rng_error = 3.0;  //Allowable range error (in yards)

    float lat_diff, lon_diff;     //The amount of Lat/Long (in Seconds
                                  //and Decimal Seconds between
                                  //Bender's current position and the
                                  //next waypoint
```

```
    float theta;                  //Angle (deg) from True North to next
                                  //waypoint
    float hdg_error;              //Angle (deg) from current heading to
                                  //next waypoint
    float new_hdg;                //The Desired heading in degrees

    double rng, temp_rng;         //Range and temporary range (in yards)

/*====================================================================*\
   Waypoint Variables
\*====================================================================*/

    typedef struct
    {
        double lat;
        double lon;
        char  action;
    }WP;                           //Define WP structure

    WP waypoints[10];              //stores the list of waypoints
    char passed_waypoint[10];      //Stores action value for passed
                                   //waypoints
    int curr_wp;                   //current wp
    char *temp;
    char *temp_lat, *temp_lon;
    char *temp_action;

    double lat, lon, wlat, wlon;


/*====================================================================*\
   Control Variables
\*====================================================================*/


    typedef struct PID
    {
        double SetPoint;

        double Proportional;
        double Integral;
        double Derivative;

        double LastError;
        double PrevError;
        double SumError;
    }PID;

    PID Servo_PID;

    //Control Flags
    int man_ctrl;
    int idle;

    //Motor Variables

    const int Motor_ch = 0;
    float Motor_spd;

    //Steering Variables (directions based on front wheel-legs)
    const float center = 1.5;
    const float max_rt = 2.0;
    const float max_lt = 1.0;
```

```
    const float half_rt = 1.75;
    const float half_lt = 1.25;
    const int Servo_ch = 1;
    float Servo_pos;            //Servo Position
    float error_value;  //hdg_error converted to value btwn -1.0 and 1.0

/*====================================================================*\
   Bus Monitoring Variables
\*====================================================================*/

    float DriveBus;
    float CtrlBus;


/*====================================================================*\
   Function Prototypes
\*====================================================================*/

    int compass_get_hdg(char sentence[MAX_SENTENCE]);

    int gps_get_position(GPSPosition *newpos, char *sentence);

    int gps_parse_coordinate(char *coord, int *degrees, float
                             *minutes);

    int ERROR_function(float new_hdg);

    int dead_reckon(void); // In development

    void msDelay (long sd);

    float PIDCalc (PID *pp, float NextPoint);


/**********************************************************************\
    Main Program
\**********************************************************************/

main()
{

    int i;
    int buf;

/*====================================================================*\
   Initialization
\*====================================================================*/

    brdInit();

    /*
     * Communications initialization
     */
    sock_init();

/*
while (ifpending(IF_DEFAULT) == IF_COMING_UP)   //debugging tool
         tcp_tick(NULL);                        //suggested by Z-world
ip_print_ifs();
*/


    if (!(host = resolve(INTERFACE_ADDRESS))) {
```

```
    exit(3);
}

if (!udp_open(&error_data, ERROR_PORT, 0xffffffff, ERROR_PORT,
    NULL)) {
    exit(3);
}
sock_mode( &error_data, TCP_MODE_ASCII);
sock_mode( &error_data, UDP_MODE_NOCHK);

if (!udp_open(&wp_data, WP_PORT, -1, WP_PORT, NULL)) {
    buf = sprintf(error_buf, "$Unable to open WP UDP session\n");
    udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
    exit(3);
}
sock_mode( &wp_data, UDP_MODE_NOCHK);

if (!udp_open(&man_data, MAN_PORT, -1, MAN_PORT, NULL)) {
    buf = sprintf(error_buf, "$Unable to open MANUAL UDP session\n");
    udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
    exit(3);
}
sock_mode( &man_data, UDP_MODE_NOCHK);

if (!udp_open(&compass_data, COMPASS_PORT, 0xffffffff, COMPASS_PORT,
    NULL)) {
    buf = sprintf(error_buf, "$Unable to open COMPASS UDP session\n");
    udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
    exit(3);
}
sock_mode( &compass_data, TCP_MODE_ASCII);
sock_mode( &compass_data, UDP_MODE_NOCHK);

if (!udp_open(&gps_data, GPS_PORT, host, GPS_PORT, NULL)) {
    buf = sprintf(error_buf, "$Unable to open GPS UDP session\n");
    udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
    exit(3);
}
sock_mode( &gps_data, TCP_MODE_ASCII);
sock_mode( &gps_data, UDP_MODE_NOCHK);

buf = sprintf(error_buf, "$Sockets are established\n");
udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
tcp_tick(NULL);

/*
 * drive init
 */
    anaOutVolts(Motor_ch, 1.5);    //set motor to stop
    anaOutVolts(Servo_ch, center);  //centers the whegs


//flags
    man_ctrl = 1;
    idle = 1;
    GPS_updated = 0;
    Compass_update = 0;
    DR_avail = 0;
    DR_nav = 0;

//compass init
    serBopen(19200);
    serBwrFlush();
```

```
        serBputs(init_str);

    //GPS init
        serCopen(9600);                //Open serial port C
        serCwrFlush();                 //Flush serial port C Buffer
        //serCputs(GPS_Parameter_Set); //Set Baud Rate (one time only)
        serCputs(GPS_Reset);           //Send Reset signal to GPS
        //serCputs(GPS_Sent_Clr);      //Send Clear signal to GPS
                                       //(one time only)
        //serCputs(GPS_GGA_Enable);    //Send GGA Sentence enable
                                       //signal (one time only)
                                       //(position info)
    //PID init
        memset (&Servo_PID,0,sizeof(PID));  //Initialize Structure
        Servo_PID.Proportional = 0.5;       //Set PID coefficients
        Servo_PID.Integral = 0.0;
        Servo_PID.Derivative = 0.0;
        Servo_PID.SetPoint = 0.0;           //Set PID Setpoint

    //Starting Message
        buf = sprintf(error_buf, "$Lopez Initiated.\nIn Manual
                              Control.\nAwaiting Orders.\n");
        udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
        tcp_tick(NULL);

/*====================================================================*\
   Primary Control Loop
\*====================================================================*/

    while (1)
    {

        /*
         * Check Bus Voltages. Critical to prevent battery damage
         */

        DriveBus = anaInVolts(5);
        CtrlBus = anaInVolts(6);
        if (DriveBus < 10)
        {
            if (DriveBus < 9.5)
            {
                buf = sprintf(error_buf, "$Drive Bus Voltage Critical. Stopping
                        Lopez!!!\n");
                udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                tcp_tick(NULL);

                anaOutVolts(Motor_ch, 1.5);
                anaOutVolts(Servo_ch, center);

                //update flags for manual control
                man_ctrl = 1;
            }
            else
            {
                buf = sprintf(error_buf, "$WARNING: Drive Bus Voltage Low.\n");
                udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                tcp_tick(NULL);
            }
        }

        if (CtrlBus < 10)
        {
```

```
            if (CtrlBus < 9.5)
            {
                buf = sprintf(error_buf, "$Control Bus Voltage Critical.
                        Stopping Lopez!!!\n");
                udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                tcp_tick(NULL);

                anaOutVolts(Motor_ch, 1.5);
                anaOutVolts(Servo_ch, center);

                //update flags for manual control
                man_ctrl = 1;
            }
            else
            {
                buf = sprintf(error_buf, "$WARNING: Control Bus Voltage
                        Low.\n");
                udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                tcp_tick(NULL);
            }
        }

/*===================================================================*\
   Begin Costatements
\*===================================================================*/

        /*
         * recieve man data
         */
        costate
        {
            waitfor(udp_recv( &man_data, cmdstr, sizeof(cmdstr))>0);

            //Tokenize the string and convert to integers
            Motor_spd = atof(strtok(cmdstr, " "));
            Servo_pos = atof(strtok(NULL, "/n"));


            //Control the motors
            anaOutVolts(Motor_ch, Motor_spd);
            anaOutVolts(Servo_ch, Servo_pos);

            if (!man_ctrl)
            {
                buf = sprintf(error_buf, "$Manual control data recieved...IN
                        MANUAL CTRL\n", curr_wp);
                udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                tcp_tick(NULL);

            }
            //Update the flags
            man_ctrl = 1;
            idle = 0;

        }//recieve man data




        /*
         * Compass
         */
```

```
        costate
        {
            waitfor ( DelayMs(compass_delay));
            serBrdFlush();
            string_pos = 0;

            input_char = serBgetc();

            //find begining of sentence
            compass_wait_time = SEC_TIMER + compass_timeout;
            //timeout if compass not working
            while (input_char != '$')
            {
                if (SEC_TIMER > compass_wait_time) abort;

                input_char = serBgetc();
                msDelay(READDELAY);
            }

            //read the sentence
            while (input_char != '*' )
            {

                    compass_sentence[string_pos] = input_char;
                    string_pos++;
                    if(string_pos == MAX_SENTENCE)
                        string_pos = 0; //reset string if too large


                input_char = serBgetc();
                //printf("%c",input_char);
                msDelay(READDELAY);
            }


            compass_sentence[string_pos] = 0; //add null
            udp_sendto(&compass_data, compass_sentence,
                        sizeof(compass_sentence), host, COMPASS_PORT);
            tcp_tick(NULL);

            if((compass_error = compass_get_hdg(compass_sentence)) != 0)
            {
                buf = sprintf(error_buf, "$Compass Error: %d\n",compass_error);
                udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                tcp_tick(NULL);
//printf("$Compass Error: %d\n %s\n",compass_error,compass_sentence);
            }
            else
            {
//printf("Current heading: %f\n", curr_hdg);
                Compass_update = 1;
            }

        //curr_hdg = 0.0;  //Leave in for testing purposes
        }// end compass costatement




        /*
         * recieve wp data
         */
```

```
        costate
        {
            waitfor(udp_recv( &wp_data, wptstr, sizeof(wptstr))>0);

            //find begining of string
            wptptr = wptstr;            //assign a pointer

            while (*wptptr != '$')      //Step to start of string
                wptptr++;

            wptptr++;

            //tokenize
            temp_lat = strtok(wptptr, " ");
            temp_lon = strtok(NULL, " ");
            temp_action = strtok(NULL, " ");

            for (i = 0; i < 10; i++)
            {
                if ((temp_lat == 0 && temp_lon ==0) ||
                    waypoints[i].action != "P")
                {
                    waypoints[i].lat = strtod(temp_lat, NULL);
                    waypoints[i].lon = strtod(temp_lon, NULL);
                    waypoints[i].action = *temp_action;
//printf("wp%d: %f %f %c\n", i, waypoints[i].lat, waypoints[i].lon,
//waypoints[i].action);
                } //End if Statement

                temp_lat = strtok(NULL, " ");
                temp_lon = strtok(NULL, " ");
                temp_action = strtok(NULL, " ");
            }//End for loop

            curr_wp = 0;  //Resets current WP to 1st waypoint. If this is
                          //an update to waypoints, Nav will increment
                          // curr_wp until a good waypoint is there.

            //update the flags
            man_ctrl = 0;
            idle = 0;

            buf = sprintf(error_buf, "$WP's recieved.
                        In AUTO NAV and preceeding to WP %d\n", curr_wp);
            udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
        }//End Waypoint Costatement




        /*
         * GPS
         */
        costate
        {
            waitfor (DelaySec(gps_delay));
            serCrdFlush();
            string_pos = 0;

            input_char = serCgetc();
            //find begining of sentence
            gps_wait_time = SEC_TIMER + gps_timeout;
```

```
    //timeout if gps not sending data
    while (input_char != '$')
    {
        if (SEC_TIMER > gps_wait_time)
        {
        gps_error_count++;
        abort;
        }
        input_char = serCgetc();
        msDelay(READDELAY);
    }

    while ((input_char != '\r') && (input_char !='\n'))
    {
        sentence[string_pos] = input_char;
        string_pos++;
        if(string_pos == MAX_SENTENCE)
            string_pos = 0; //reset string if too large

        input_char = serCgetc();
        msDelay(READDELAY);
    }

    sentence[string_pos] = 0;

    udp_sendto(&gps_data, sentence, sizeof(sentence), host,
                GPS_PORT);
    //tcp_tick(NULL);

    gps_error = gps_get_position(&current_pos, sentence);
        /*     GPS ERROR CODES
             0 - successful
            -1 - not differential
            -2 - sentence marked invalid
            -3 - parsing error
         */
    if ((gps_error == 0) || (gps_error == -1))
    {
        gps_error_count = 0;
        GPS_updated = 1;
        DR_avail = 1;
        Vx = 0, Vy = 0, Vz = 0; //resets velocity values when GPS
                                //is restored
        curr_lat=(current_pos.lat_degrees +
                    (current_pos.lat_minutes/60));
        curr_lon=(current_pos.lon_degrees +
                    (current_pos.lon_minutes/60));
    }
    else
    {
        gps_error_count ++;

        //If GPS gives bad data for 6 tries, the program will
        //go to dead reckoning unless no initial GPS fix has
        //been achieved. In this case, Lopez stops.
    }
}//End GPS costatement




/*
```

```
         * DR
         */
        costate
        {
            if ((gps_error_count > 6) && (!man_ctrl))
                {
                    buf = sprintf(error_buf, "$GPS error count exceeded.\nDR
                                    unavailable.\nLopez in MANUAL CONTROL.\n");
                    udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                    tcp_tick(NULL);
                    anaOutVolts(Motor_ch, 1.5);
                    anaOutVolts(Servo_ch, center);

                    man_ctrl = 1;   //update flag for manual control
                    abort;
                }
                else
                {
                    buf = sprintf(error_buf, “$Loss of GPS.\nSwitching to
                                                DR.\n”);
                    udp_sendto(&error_data, error_buf, buf, host, ERROR_PORT);
                    tcp_tick(NULL);
                    DR_nav = dead_reckon();
                    curr_lat+=DR_lat_adj;
                    curr_lon+=DR_lon_adj;

                    lat_mant=modf(curr_lat,&lat_int);
                    lon_mant=modf(curr_lon,&lon_int);
                    ftoa(lat_int, lat_int_ascii);
                    ftoa(lat_mant, lat_mant_ascii);
                    ftoa(lon_int, lon_int_ascii);
                    ftoa(lon_mant,lon_mant_ascii);
                    strcat(lat_str,lat_int_ascii);
                    strcat(lat_str,”.”);
                    strncat(lat_str, lat_mant_ascii,4);
                    strcat(lon_str,lon_int_ascii);
                    strcat(lon_str,”.”);
                    strncat(lon_str, lon_mant_ascii,4);

                    buf = sprintf(error_buf,
                    “$GPGGA,000000,%s,N,%s,W,0,00,,,,,\n\r:,lat_str,lont_str);
                    udp_sendto(&gps_data, error_buf, buf, host, GPS_PORT);
                    tcp_tick(NULL);
                }
            }
        }//End DR Costatement



        /*        **** Passes heading error and range to CTRL
         * Nav    **** costatement and uses error function to
         *        **** find error from new_hdg and curr_heading
         */
            costate
            {
            if (man_ctrl) abort;        //Do not do anything if in
                                        //manual control

            if ((GPS_updated) || (DR_nav))//Navigates to new waypoint
            {
//if(1)             {
// give fake lat/long
```

```
//curr_lat = 36.595;        //For Testing
//curr_lon = 121.8753;      //For Testing

                lat = 60 * curr_lat;      //converts latitude into
                                          //Minutes and decimal minutes.
                lon = 60 * curr_lon;      //converts longitude into
                                          //Minutes and decimal minutes.
                wlat = 60 * waypoints[curr_wp].lat;  //Converts waypoint
                                                     //values to
                wlon = 60 * waypoints[curr_wp].lon;  //decimal minutes.

                rng = sqrt((((2000 * wlat) - (2000 * lat)) * ((2000 *
                    wlat) -(2000 * lat))) + (((1600 * wlon) - (1600 *
                    lon)) *((1600 * wlon) - (1600 * lon))));

                if (rng < rng_error)  //When close enough to waypoint,
                                      //action code takes effect and next
                                      //waypoint is loaded
                {

                    switch (waypoints[curr_wp].action)
                    {
                        case 'T':               //Go to next waypoint
                        {
                            passed_waypoint[curr_wp] = 'T';
                            //Stores action code in temp array
                            waypoints[curr_wp].action = 'P';
                            //Changes action code to indicate WP has been
                            //passed
                            curr_wp++;
                            buf=sprintf(error_buf, "$Passed WP %d\n”, curr_wp);
                            udp_sendto(&error_data, error_buf, buf, host,
                                        ERROR_PORT);


                            while ((waypoints[curr_wp].lat == 0) &&
                                (waypoints[curr_wp].lon == 0))
                            {                 //checks for valid WP
                                curr_wp++;

                                if (curr_wp == 10)
                                {
                                    buf=sprintf(error_buf, "$No Valid WP
                                            Found.\n");
                                    udp_sendto(&error_data, error_buf,
                                                buf, host, ERROR_PORT);
                                    tcp_tick(NULL);
                                    man_ctrl = 1;
                                    abort;
                                    }//End if
                                }//End while

                                 break;
                             }//End case 'T'

                            case 'H':          //Start from beginning again
                            {
                                for (i = 0;i < 10;i++)
                                //Reloads prior action codes
                                {
                                    waypoints[i].action = passed_waypoint[i];
                                }
```

```
    curr_wp = 0;

    while ((waypoints[curr_wp].lat == 0) &&
        (waypoints[curr_wp].lon == 0))
    {                   //checks for valid WP
        curr_wp++;
        if (curr_wp == 10)
        {
            buf=sprintf(error_buf, "$No Valid WP
                    Found\n");
            udp_sendto(&error_data, error_buf,
                       buf, host, ERROR_PORT);
            tcp_tick(NULL);
            man_ctrl = 1;
            abort;
        }//End if
    }//End while

    break;
}//End case 'H'

case 'S':              //Stop
{
    anaOutVolts(Motor_ch, 1.5);
    anaOutVolts(Servo_ch, center);   //Stops Lopez

    for (i = 0;i < 10;i++) //Clears Waypoint array
    {
        waypoints[i].lat = 0;
        waypoints[i].lon = 0;
        waypoints[i].action='T';
    }//End for loop

    buf=sprintf(error_buf, "$Destination Achieved,\n
            Waypoints cleared.\n");
    udp_sendto(&error_data, error_buf,
                buf, host, ERROR_PORT);
    tcp_tick(NULL);
    man_ctrl = 1;
    idle = 1;
    abort;
}//End case 'S'

case 'P':          //Check for passed waypoints
{
    curr_wp++;
    //Lopez ignores this point and goes to next
    while ((waypoints[curr_wp].lat == 0) &&
        (waypoints[curr_wp].lon==0))
    {                   //checks for valid WP
        curr_wp++;
        if (curr_wp == 10)
        {
            buf=sprintf(error_buf, "$No Valid WP
                    Found\n");
            udp_sendto(&error_data, error_buf,
                       buf, host, ERROR_PORT);
            tcp_tick(NULL);
            man_ctrl = 1;
            abort;
        }//End if
    }//End while
```

```
                        break;
                        }//End case 'P'

                        default: //Indicates invalid action
                        {
                            buf =sprintf(error_buf, "$Invalid action for WP #
                                %d\n", curr_wp);
                            udp_sendto(&error_data, error_buf,
                                buf, host, ERROR_PORT);
                            tcp_tick(NULL);

                            anaOutVolts(Motor_ch, 1.5);    //Stops Lopez and
                            anaOutVolts(Servo_ch, center); // places in manual
                            man_ctrl = 1;                  //control
                            idle = 1;
                            abort;


                        }//End default case
                    }//End Switch

                    if (curr_wp > 9) //Should not be here.
                                     //Action for last WP invalid.
                    {
                        anaOutVolts(Motor_ch, 1.5);    //Stops Lopez and places
                        anaOutVolts(Servo_ch, center); //in manual control
                        man_ctrl = 1;

                        buf=sprintf(error_buf, "$Invalid action for wp 9\n");
                        udp_sendto(&error_data, error_buf, buf,
                                   host,ERROR_PORT);
                        tcp_tick(NULL);
                        abort;
                     }//End if (curr_wp>9)

                }//End if (rng < rng_error)

                //If range not within error, calculate new heading.
                // 3600 converts lat_diff and lon_diff to decimal
                // seconds for accuracy
                lat_diff = 3600 * (waypoints[curr_wp].lat-curr_lat);
                lon_diff = 3600 * (curr_lon –
                    waypoints[curr_wp].lon);
//printf("wp0_lat:  %g\twp0_lon: %g\n", waypoints[curr_wp].lat,
//waypoints[curr_wp].lon);
//printf("lat_diff: %g\tlon_diff: %g\n", lat_diff, lon_diff);

                // determine theta in degrees
                theta = atan((lat_diff) / (lon_diff)) * (180 / pi);
//printf("theta: %g\n", theta);

                // waypoint located in positive y-axis
                if ((lon_diff == 0) && (lat_diff > 0))
                    new_hdg = 0;

                //waypoint is located in negative y-axis
                else if ((lon_diff == 0) && (lat_diff < 0))
                    new_hdg = 180;

                //waypoint is located in positive x-axis
                else if ((lon_diff > 0) && (lat_diff == 0))
                    new_hdg = 90;
```

```
                //waypoint is located in negative x-axis
                else if ((lon_diff < 0) && (lat_diff == 0))
                    new_hdg = 270;

                //waypoint is located in the first or fourth quadrant
                //(0-90 or 270-0)
                else if ((lon_diff > 0) && (lat_diff != 0))
                    new_hdg = 90 - theta;

                //waypoint is located in the second or third quadrant
                //(90-180 or 180-270)
                else if ((lon_diff < 0) && (lat_diff != 0))
                    new_hdg = 270-theta;

                hdg_error = ERROR_function(new_hdg);
//printf("cur_hdg:  %g\tnew_hdg:  %g\t  hdg_err_nav:  %g\n",  curr_hdg,  new_hdg,
//hdg_error);
            }// End if (GPS_updated)

            }//End nav costate


        /*
         *   Lopez Control
         */

        costate
        {
            waitfor(!man_ctrl);

            //steering
            if ((fabs(hdg_error) >= brg_error) && (rng > rng_error))
            {
                if (hdg_error < -90)    //Any heading error greater than
                    hdg_error = -90;    //90 degrees will need the max turn,
                if (hdg_error > 90)     //so values greater than this
                    hdg_error = 90;     //are clipped.
                error_value=(heg_error/90); //Coverts hdg_error to value btwn
                                            //-1.0 and 1.0
                Servo_pos = PIDCalc(&Servo_PID, error_value)+1.5;
                if (Servo_pos < 1)
                    Servo_pos = 1;
                if (Servo_pos > 2)
                    Servo_pos = 2;
}
            //speed
            if (rng >= 20) Motor_spd = 2.0;
            if ((rng <20) && (rng >= 10)) Motor_spd = 1.75;

            anaOutVolts(Servo_ch, Servo_pos);
            anaOutVolts(Motor_ch, Motor_spd);
        }//End Control Costatement
    } //End while(1)
}//End of main


////////////////////////////////////////////////////////////////


/*START FUNCTION DESCRIPTION*****************************************
compass_get_hdg

SYNTAX: int compass_get_data();
```

```
KEYWORDS:      compass

DESCRIPTION:    Parses a sentence to extract heading data.
This function is able to parse HPR data from a HMR3000 Digital Compass

PARAMETER1: sentence - a string containing a line of HPR data

RETURN VALUE:    0 - success
        -1 - parsing error
        -2 - heading marked invalid

SEE ALSO:

END DESCRIPTION **********************************************************/

int compass_get_hdg(char sentence[MAX_SENTENCE])
{
    auto int i;
    char *err,*hdg,*type;
    char *pitch, * pitch_error, *roll, *roll_error;
    char error;

    if(strlen(sentence) < 4)
        return -1;
    if(strncmp(sentence, "$PTNTHPR", 8) == 0)
    {
        //parse hpr sentence
        type = strtok(sentence, ",");
        hdg = strtok(NULL, ",");
        err = strtok (NULL, ",");
        pitch = strtok(NULL, ",");
        pitch_error = strtok(NULL, ",");
        roll = strtok(NULL, ",");
        roll_error = strtok(NULL, ",");
//NOTE in order to turn compass axis off fwd axis of robot, pitch and roll will
//have to be swapped and signs may need adjustment in imbedded and Java Code
        if(hdg == NULL)
            return -2;

        //pull out data
        curr_hdg = atof(hdg);
        curr_pitch = -(atof(roll));
        curr_roll = atof(pitch);

        error = (int)err;
        if (strncmp(&error, "N", 1) == 0)
            return -2;

    }
    else
        return -1;

    return 0;

}

/*START FUNCTION DESCRIPTION*****************************************
gps_parse_coordinate

SYNTAX: gps_parse_coordinate(char *coord, int *degrees, float *minutes)

KEYWORDS:      gps parse
```

```
DESCRIPTION:    Parses GPS position data

PARAMETER1: coord - contains N/S, E/W
            degrees, minutes - positional information

RETURN VALUE:    0 - success (xxxxx.xxxx minutes)
            -1 - parsing error

SEE ALSO:

END DESCRIPTION **********************************************************/

nodebug int gps_parse_coordinate(char *coord, int *degrees, float *minutes)
{
    auto char *decimal_point;
    auto char temp;
    auto char *dummy;

    decimal_point = strchr(coord, '.');
    if(decimal_point == NULL)
        return -1;
    temp = *(decimal_point - 2);
    *(decimal_point - 2) = 0; //temporary terminator
    *degrees = atoi(coord);
    *(decimal_point - 2) = temp; //reinstate character
    *minutes = strtod(decimal_point - 2, &dummy);
    return 0;
}


/* START FUNCTION DESCRIPTION ***************************************
gps_get_position

SYNTAX:         int gps_get_position(GPSPositon *newpos, char *sentence);

KEYWORDS:       gps

DESCRIPTION:    Parses a sentence to extract position data.
This function is able to parse any of the following GPS sentence formats: GGA

PARAMETER1:     newpos - a GPSPosition structure to fill
PARAMETER2:     sentence - a string containing a line of GPS data
                    in NMEA-0183 format

RETURN VALUE:    0 - success
            -1 - not differential
            -2 - sentence marked invalid
            -3 - parsing error

SEE ALSO:

END DESCRIPTION **********************************************************/

nodebug int gps_get_position(GPSPosition *newpos, char *sentence)
{
    auto int i;

    if(strlen(sentence) < 4)
        return -3;
    if(strncmp(sentence, "$GPGGA", 6) == 0)
    {
        //parse GGA sentence
```

```
        for(i = 0;i < 11;i++)
        {
            sentence = strchr(sentence, ',');
            if(sentence == NULL)
                return -3;
            sentence++; //first character in field
            //pull out data
            if(i == 1) //latitude
            {
                if( gps_parse_coordinate(sentence,
                        &newpos->lat_degrees,
                        &newpos->lat_minutes)
                  )
                {
                    return -3; //get_coordinate failed
                }
            }
            if(i == 2) //lat direction
            {
                newpos->lat_direction = *sentence;
            }
            if(i == 3) // longitude
            {
                if( gps_parse_coordinate(sentence,
                        &newpos->lon_degrees,
                        &newpos->lon_minutes)
                  )
                {
                    return -3; //get_coordinate failed
                }
            }
            if(i == 4) //lon direction
            {
                newpos->lon_direction = *sentence;
            }
            if(i == 5) //link quality
            {
                if(*sentence == '0')
                    return -2;
                if(*sentence == '1')
                    return -1;
            }
        }
    }
    else
    {
        return -3; //unknown sentence type
    }
    return 0;
}


/*START FUNCTION DESCRIPTION*****************************************
ERROR_function

SYNTAX:         int ERROR_function(new_hdg);

KEYWORDS:       nav, control

DESCRIPTION:    Determines heading error for use by Nav and Control
costatements

PARAMETER1:     new_hdg - latest update of bearing to next waypoint
```

```
                or direction to drive based upon sonar contact

RETURN VALUE:   hdg_error

SEE ALSO:

END DESCRIPTION **********************************************************/
int ERROR_function(float new_hdg)
{
    hdg_error = (360 + new_hdg) - curr_hdg;

    if (hdg_error > 360)
        hdg_error = hdg_error - 360;

    if(hdg_error > 180)
        hdg_error = hdg_error - 360;

    return((int)hdg_error);
}


/*START FUNCTION DESCRIPTION*********************************************
gps_get_position

SYNTAX:     void msDelay(long sd);

KEYWORDS:   delay, wait

DESCRIPTION:    introduces a defined ms delay loop

PARAMETER1:     sd - number of ms to wait

SEE ALSO:

END DESCRIPTION **********************************************************/
void msDelay (long sd)
{
    unsigned long t1;

    t1 = MS_TIMER;
    for (t1 = MS_TIMER; MS_TIMER < (sd + t1); );
}


/*START FUNCTION DESCRIPTION*********************************************
dead_reckon

SYNTAX: int dead_reckon(void)

KEYWORDS: nav, dead reckon, estimate

DESCRIPTION:    calculates changes in latitude and longitude by a double
integration of voltage levels from a CROSSBOW MEMS accelerometer that is
normalized to the gravitational field and adjusted to magnetic north through
inputs from the compass.

PARAMETERS: None

RETURN VALUE: 1

SEE ALSO:

END DESCRIPTION**********************************************************/
```

```
int dead_reckon(void)
{
    //Variables
    auto float Axc, Ayc, Azc;
        //Acceleration calibration values (2.5V = 0g)
    auto float Ax, Ay, Az;//calculated acceleration values in yd/s^2
    auto float Dx, Dy, Dz;//calculated distances in platform F.O.R.
    auto float Dfwd, Dside;
        //calculated distances (normal to Earth's   gravity)
    auto float Dlat, Dlong;//calculated distances in geodetic F.O.R.
    auto int dt, last_time; //time management variables
    int X_chan, Y_chan, Z_chan;//Analog input channels for three axis

    dt= MS_Timer - Last_Time;
    Ax = (float anaInVolts(X_chan)-Axc)*(10.72/.2);
        //10.72 is g in yd/s^2
    Ay = (float anaInVolts(Y_chan)-Ayc)*(10.72/.2);
        //.2 is 200mV/g (parameter of accelerometer design)
    Az = (float anaInVolts(Z_chan)-Azc)*(10.72/.2);

    Dx = (Vx + Ax*(dt/1000))*(dt/1000);
    Dy = (Vy + Ay*(dt/1000))*(dt/1000);
    Dz = (Vz + Az*(dt/1000))*(dt/1000);

    Vx+=(Vx + Ax*(dt/1000));//reset velocity values
    Vy+=(Vx + Ax*(dt/1000));
    Vz+=(Vx + Ax*(dt/1000));

    //perform pitch and roll calculations

    Dfwd = (Dx*cos(curr_pitch*(pi/180)))-(Dz*sin(curr_pitch*(pi/180)));
        //pitch measured positive up
    Dside = (Dy*cos(curr_roll*(pi/180)))-(Dz*sin(curr_roll*(pi/180)));
        //roll measured from port side positive up

    //adjust for heading
    Dlat = Dfwd*cos(curr_hdg*(pi/180))+Dside*sin(curr_hdg*(pi/180));
    Dlong = Dfwd*sin(curr_hdg*(pi/180))-Dside*cos(curr_hdg*(pi/180));

    DR_lat_adj = Dlat/120000;    //distance moved in decimal degrees
    DR_long_adj = Dlong/96000;
    last_time=MS_Timer;
    return 1;
}


/*START FUNCTION DESCRIPTION*****************************************
PIDCalc

SYNTAX:     double PIDCalc (PID *pp, double NextPoint)

KEYWORDS:   Control, Proportional, integral, Derivative

DESCRIPTION:    this function is a modified version of a prototype function
written by Greg Young at Z-world.

PARAMETERS:     *pp is a pointer to PID structure
                NextPoint is a value between -1 and 1 determined from hdg_error

RETURN VALUE:   Returns a value for Servo positioning

SEE ALSO:
```

```
END DESCRIPTION **********************************************************/
float PIDCalc (PID *pp, float NextPoint)
{
    float dError, Error;

    pp->SumError += (Error = pp->SetPoint - NextPoint);
    dError - pp->LastError - pp->PrevError;
    pp->PrevError - pp->LastError;
    pp->LastError = Erro;
    return (pp->Proportional * Error +
            pp->Integral * pp->SumError +
            pp->Derivative * dError);
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. JAVA CODE

```
//---------------------------------------------------------------------
// Filename:   LopezMainApp.java
// Author:     Kubilay Uzun
// Date:       9/13/2003
// Modified:   Jason Ward
// Project:    Lopez
// Compiler:   JDK 1.4.1_02
//---------------------------------------------------------------------

import javax.swing.*;
import javax.imageio.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

/**
 * The MainApp class provides all storage and services in order to
 * monitor and control the robot. The mission structure including
 * waypoints and actions are constructed and uploaded to the robot by
 * this class. With a user friendly GUI interface, MainApp helps the
 * mission to be costructed on a detailed map of the area. In addition,
 * all robot sensor data can be monitored continously via MainApp. Manual
 * control of robot is accomplished by a mouse-drag joystick which is
 * very natural to adapt. All network communications performed by MainApp
 * use Datagram connections which are fast and not sensitive to fading
 * and connection losses which are highly probable in a wireless network
 * environment.
 *
 * @author Kubilay Uzun
 */
public class MainApp extends JFrame
                     implements  MouseMotionListener,
                                 MouseListener,
                                 WindowListener,
                                 ActionListener {
    /**
     * MOTOR_MIN,MOTOR_MAX, SERVO_MIN and SERVO_MAX constants set the
     * precision of manual control inputs. Selecting a small or big number
     * is a trade-off between precision and response speed. Since the
     * control outputs are not sent unless they're changed small numbers
     * cause less control packets to be transmitted. However, Datagram
     * connection used for this task is so fast that usually there is no
     * reason to worry about overloading buffer and degrading the control
     * response.
     */
    static final double MOTOR_MIN = 0;
    static final double MOTOR_MAX = 100;
    static final double SERVO_MIN = 0;
    static final double SERVO_MAX = 100;

    /**
     * MOTOR_MIN_VOLTS, MOTOR_MAX_VOLTS, SERVO_MIN_VOLTS and
     * SERVO_MAX_VOLTS constants are set according to the voltage specs of
     * the pulse-width modulator (in Volts).
     */
```

```
static final double MOTOR_MIN_VOLTS = 1;
static final double MOTOR_MAX_VOLTS = 2;
static final double SERVO_MIN_VOLTS = 1;
static final double SERVO_MAX_VOLTS = 2;
private double Motor = (MOTOR_MAX + MOTOR_MIN) / 2;
private double Servo = (SERVO_MAX + SERVO_MIN) / 2;
private double MotorOld, ServoOld;

/**
 * Video mode constants and globals.
 */
static final int NAV_MAP = 0;
static final int VIS_CAM = 1;
static final int IR_CAM = 2;
static final int VIS_COLUMNS = 80;
static final int VIS_ROWS = 143;
static final double VIS_PIXEL_WIDTH = 8;
static final double VIS_PIXEL_HEIGHT = 4; //For asp. ratio
static final int MAX_CAM_BYTES = 80000;

double visCurX = VIS_COLUMNS * VIS_PIXEL_WIDTH;
double visCurY = VIS_ROWS * VIS_PIXEL_HEIGHT;
int rgb;
private int videoMode = NAV_MAP;

String camStream;

/**
 * Map constants and globals. NAVMAP_FILE_NAME sets the file name
 * to be used as an area map. This should be a JPEG file. Huge files
 * need extended times to be loaded. Since the map is periodically
 * updated, using huge files may seriously degrade the performance of
 * the program. Files up to 1MByte are tolerated well depending on the
 * performance of the platform.
 */
static final String NAVMAP_FILE_NAME = "NavMap_small.jpg";
private JLabel map;
private Graphics mapGraph;
private Image img;
private ImageIcon mapIcon;
private double WptCircleDia = 20; //WayPoint circle diameter

/**
 * Global GUI items. These items are used by various methods of the
 * program for I/O.
 */
private JButton joyStickPanel, scalingButton, insertWpt;
private JTextArea messageArea;
private JLabel cursorCoord;
private JComboBox wayPointField;
private JTextField fixTimeField,
                   numberOfSatellitesField,
                   gpsField,
                   latField,
                   lonField,
                   headField,
                   pitchField,
                   rollField,
                   netField,
                   navModeField;

/**
 * oldErrorMessage is used to keep track of the Lopez origined error
```

```
 * messages are being changed or not. Since the communication is
 * asynchronous, this prevents the same message from being displayed
 * repeatedly if there's nothing new.
 */
private String oldErrorMessage;

/**
 * Networking constants and globals. Robot IP adress is defined here.
 * This implies that a compiled intance of MainApp belongs to a certain
 * robot.In order to control other platforms ROBOT_IP_ADRESS should be
 * modified.
 */
static final String ROBOT_IP_ADRESS = "192.168.0.91";
static final String CAM_IP_ADRESS = "192.168.0.92";

/**
 * Ports. Starting from 401, ports are used for Manual control,
 * route upload, gps monitor, compass monitor and Lopez error message
 * monitor respectively.
 */
static final int CMD_PORT = 4001;
static final int ROUTE_PORT = 4002;
static final int GPS_PORT = 4003;
static final int COMPASS_PORT = 4004;
static final int ERROR_PORT = 4005;
static final int CAM_PORT = 4006;

static final int CMD_BUFFER_SIZE = 2048;
static final int ROUTE_BUFFER_SIZE = 2048;

/**
 * Datagram sockets. Note here that sockets are defined only for data
 * output. The reason for this is data output is non-blocking. So,
 * there no reason for using separate threads for Datagram output.
 */
private DatagramSocket cmdSock;
private DatagramSocket routeSock;

/**
 * Datagram input threads. The blocking nature of asynchronous Datagram
 * input pushes us to create separate threads for each input socket.
 * See PacketReceivingThread for detailed explanation.
 */
private PacketReceivingThread gpsThread;
private PacketReceivingThread compassThread;
private PacketReceivingThread errorThread;
private PacketReceivingThread camThread;

/**
 * Timers. In this program javax.swing.Timer class is used. What is the
 * reason for an initial delay? Since the timer is started immediately
 * after timer.start() method is called, the tasks which are performed
 * by the timer event may be calling not-yet-constructed objects. This
 * probably causes the timer event to return a nullPointerException.
 * Initial delay should ensure fully construction of the frame before
.* operating.
 */
static final int TMR_INITIAL_DELAY = 10000;
static final int TMR1_DELAY = 1000;
private javax.swing.Timer tmr1;

/**
 * Waypoints. Maximum number of waypoints per mission is set by
```

```
 * MAX_WAYPOINTS constant. To change this, MAX_WAYPOINTS constant
 * should be modified.
 */
static final int MAX_WAYPOINTS = 10;
static final int INSERT = 0;
static final int OVERRIDE = 1;

private int currentWpt;
private int wptInsertMode;
private int nowScaling;
private double scalePos1Lat, scalePos1Lon, scalePos2Lat, scalePos2Lon;
private double scalePos1x, scalePos1y, scalePos2x, scalePos2y;
private double scaleX, scaleY;

private WayPoint route[] = new WayPoint[MAX_WAYPOINTS];
private WayPoint scalingWpt1;
private WayPoint scalingWpt2;
private WayPoint mapCursorWpt;

private NewWayPoint nwpt;

/**
 * Robot position. Current robot position and heading is retrieved from
 * the Datagram socket, tokenized, parsed and assigned to these data
 * members. Constant ROBOT_HEADING_TICK_LENGTH should be modified to
 * change the heading tick.
 */
static final int ROBOT_HEADING_TICK_LENGTH = 20;
private WayPoint robotPos;
private double robotHeading;

/**
 * File streams. Scaling data of the current mission map is read and
 * written each time program is entered or exited. Route streams
 * provide load/save functions of the current route.
 */
private FileInputStream fStorIn;
private FileOutputStream fStorOut;
private DataInputStream storIn;
private DataOutputStream storOut;
private FileInputStream fRouteIn;
private FileOutputStream fRouteOut;
private ObjectInputStream routeIn;
private ObjectOutputStream routeOut;

/**
 * The MainApp() constructor initializes data members of the main
 * application and creates all GUI elements.
 *
 * @param none
 * @exception none
 */
public MainApp() {
    //Application's name
    super("LOPEZ CONTROL INTERFACE");

    //Size. Somehow it does not maximize???
    setSize(new Dimension(800,600));
    setExtendedState(Frame.MAXIMIZED_BOTH);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    //Start javax.swing.Timer
    tmr1 = new javax.swing.Timer(TMR1_DELAY, this);
```

```
tmr1.setInitialDelay(TMR_INITIAL_DELAY);
tmr1.start();

//Initialize current robot pos/heading to default
try {
robotPos = new WayPoint("N0 0.0 W0 0.0", "Turn");
robotHeading = 0;
} catch (Exception ex) {}

//Create an istance of NewWayPoint
nwpt = new NewWayPoint(this);

//main panel
JPanel pnl1 = new JPanel(new BorderLayout());
setContentPane(pnl1);
addWindowListener(this);

//Center Video Panel
JPanel ctrPanel = new JPanel(new BorderLayout());
pnl1.add(ctrPanel, BorderLayout.CENTER);
JPanel ctrButtonPanel = new JPanel(new GridLayout(1,3));
ctrPanel.add(ctrButtonPanel, BorderLayout.SOUTH);
JButton mapButton = new JButton("NAVIGATION MAP");
ctrButtonPanel.add(mapButton);
mapButton.addActionListener(this);
JButton visButton = new JButton("VISUAL CAM");
ctrButtonPanel.add(visButton);
visButton.addActionListener(this);
JButton irButton = new JButton("THERMAL CAM");
ctrButtonPanel.add(irButton);
JPanel ctrUpperButtonPanel = new JPanel(new GridLayout(1,3));
ctrPanel.add(ctrUpperButtonPanel, BorderLayout.NORTH);
scalingButton = new JButton("PERFORM MAP SCALING");
scalingButton.addActionListener(this);
ctrUpperButtonPanel.add(scalingButton);
JButton cup2Button = new JButton();
ctrUpperButtonPanel.add(cup2Button);
cursorCoord = new JLabel("N0 0.0 W0 0.0");
ctrUpperButtonPanel.add(cursorCoord);

//Mission map
map = new JLabel();
//Put this into a JScrollPane
JScrollPane mapPane = new JScrollPane(map);
ctrPanel.add(mapPane, BorderLayout.CENTER);
//Name is used to discriminate between GUI element calls
map.setName("M");
//Change cursor for accuracy of perception
map.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
map.addMouseListener(this);
map.addMouseMotionListener(this);

//Read map from file
File file = new File(NAVMAP_FILE_NAME);
try {
    img = ImageIO.read(file);
} catch (Exception ex) {
    messageArea.append("\nCannot Read Navigation Map.\n");
}

mapIcon = new ImageIcon(img);
map.setIcon(mapIcon);
```

```
//Manual Control & Joystick Panels
JPanel pnl2 = new JPanel(new GridLayout(3,1));
pnl1.add(pnl2,BorderLayout.EAST);

JPanel buttonControlPanel = new JPanel(new BorderLayout());
pnl2.add(buttonControlPanel);

joyStickPanel = new JButton("JOYSTICK");
//Name is used to discriminate between GUI element calls
joyStickPanel.setName("J");
pnl2.add(joyStickPanel);
joyStickPanel.addMouseMotionListener(this);

JButton stopButton = new JButton("STOP");
buttonControlPanel.add(stopButton, BorderLayout.CENTER);
stopButton.addActionListener(this);
JButton fwdButton = new JButton("FORWARD");
buttonControlPanel.add(fwdButton, BorderLayout.NORTH);
fwdButton.addActionListener(this);
JButton rvsButton = new JButton("REVERSE");
buttonControlPanel.add(rvsButton, BorderLayout.SOUTH);
rvsButton.addActionListener(this);
JButton rightButton = new JButton("RIGHT");
buttonControlPanel.add(rightButton, BorderLayout.EAST);
rightButton.addActionListener(this);
JButton leftButton = new JButton("LEFT");
buttonControlPanel.add(leftButton, BorderLayout.WEST);
leftButton.addActionListener(this);

//Position & Waypoint Panels
JPanel pnl3 = new JPanel(new GridLayout(3,1));
pnl1.add(pnl3,BorderLayout.WEST);

JPanel positionPanel = new JPanel(new GridLayout(9,2));
pnl3.add(positionPanel);
JLabel fixTimeLabel = new JLabel("LAST FIX TIME");
positionPanel.add(fixTimeLabel);
fixTimeField = new JTextField();
positionPanel.add(fixTimeField);
JLabel numberOfSatellitesLabel = new JLabel("NUMBER of SATELLITES");
positionPanel.add(numberOfSatellitesLabel);
numberOfSatellitesField = new JTextField();
positionPanel.add(numberOfSatellitesField);
JLabel gpsLabel = new JLabel("GPS STATUS");
positionPanel.add(gpsLabel);
gpsField = new JTextField();
positionPanel.add(gpsField);
JLabel latLabel = new JLabel("LAT");
positionPanel.add(latLabel);
latField = new JTextField();
positionPanel.add(latField);
JLabel lonLabel = new JLabel("LON");
positionPanel.add(lonLabel);
lonField = new JTextField();
positionPanel.add(lonField);
JLabel headLabel = new JLabel("HEADING");
positionPanel.add(headLabel);
headField = new JTextField();
positionPanel.add(headField);
JLabel pitchLabel = new JLabel("PITCH");
positionPanel.add(pitchLabel);
pitchField = new JTextField();
positionPanel.add(pitchField);
```

```
    JLabel rollLabel = new JLabel("ROLL");
    positionPanel.add(rollLabel);
    rollField = new JTextField();
    positionPanel.add(rollField);
    JLabel netLabel = new JLabel("NETWORK STATUS");
    positionPanel.add(netLabel);
    netField = new JTextField("Datagram");
    positionPanel.add(netField);

    JPanel wayPointPanel = new JPanel(new GridLayout(6,2));
    pnl3.add(wayPointPanel);

    JButton loadRoute = new JButton("LOAD ROUTE");
    wayPointPanel.add(loadRoute);
    loadRoute.addActionListener(this);
    JButton saveRoute = new JButton("SAVE ROUTE");
    wayPointPanel.add(saveRoute);
    saveRoute.addActionListener(this);
    JLabel navModeLabel = new JLabel("NAV MODE");
    wayPointPanel.add(navModeLabel);
    navModeField = new JTextField("MANUAL");
    wayPointPanel.add(navModeField);
    JLabel wayPointLabel = new JLabel("WAYPOINT");
    wayPointPanel.add(wayPointLabel);
    wayPointField = new JComboBox();
    Integer ii;
    for (int i = 1; i <= MAX_WAYPOINTS; i++) {
        ii = new Integer(i);
        wayPointField.addItem(ii.toString());
    }
    wayPointField.addActionListener(this);
    wayPointPanel.add(wayPointField);
    JButton wptDecrease = new JButton("<");
    wayPointPanel.add(wptDecrease);
    wptDecrease.addActionListener(this);
    JButton wptIncrease = new JButton(">");
    wayPointPanel.add(wptIncrease);
    wptIncrease.addActionListener(this);
    insertWpt = new JButton("Inserting");
    wayPointPanel.add(insertWpt);
    insertWpt.addActionListener(this);
    JButton deleteWpt = new JButton("DELETE WAYPOINT");
    wayPointPanel.add(deleteWpt);
    deleteWpt.addActionListener(this);
    JButton editWpt = new JButton("EDIT WAYPOINT");
    wayPointPanel.add(editWpt);
    editWpt.addActionListener(this);
    JButton sendRoute = new JButton("SEND ROUTE");
    wayPointPanel.add(sendRoute);
    sendRoute.addActionListener(this);

    //Message Output Area
    messageArea = new JTextArea(20,22);
    JScrollPane masp = new JScrollPane(messageArea);
    pnl3.add(masp);

    //Construct route
    for (int i = 0; i < MAX_WAYPOINTS; i++) {
        route [i] = new WayPoint();
    }
} //Constructor

/**
```

```
 * Establish all Datagram connections/instantiate and start all packet
 * receiving threads.
 */
public void establishDatagramConnections() {
    messageArea.append("\nConnecting to Datagram Sockets...");

    InetAddress hostAdress= null;
    try {
        //Instantiate sockets for data output
        cmdSock = new DatagramSocket();
        cmdSock.connect(hostAdress.getByName(ROBOT_IP_ADRESS), CMD_PORT);
        cmdSock.setSendBufferSize(CMD_BUFFER_SIZE);

        routeSock = new DatagramSocket();
        routeSock.connect(hostAdress.getByName(ROBOT_IP_ADRESS),
                                               ROUTE_PORT);
        routeSock.setSendBufferSize(ROUTE_BUFFER_SIZE);

        //Instantiate threads for data input
        gpsThread = new PacketReceivingThread(ROBOT_IP_ADRESS,GPS_PORT);
        compassThread = new PacketReceivingThread(ROBOT_IP_ADRESS,
                                                  COMPASS_PORT);
        errorThread = new PacketReceivingThread(ROBOT_IP_ADRESS,
                                                ERROR_PORT);
        camThread = new PacketReceivingThread(CAM_IP_ADRESS,CAM_PORT);
        //Start threads
        gpsThread.start();
        compassThread.start();
        errorThread.start();
        camThread.start();
    } catch (Exception ex) {
        messageArea.append("Failed.\n");
        return;
    }
    messageArea.append("Success.\n");
}//establishDatagramConnections

/**
 * Send control voltages to the control socket
 */
public void sendControlData() {
    double MotorVolts = 0;
    double ServoVolts = 0;

    //Calculate motor and servo voltages
    MotorVolts = MOTOR_MIN_VOLTS +
                ((Motor/ (MOTOR_MAX - MOTOR_MIN)) *
                (MOTOR_MAX_VOLTS - MOTOR_MIN_VOLTS ));
    ServoVolts = MOTOR_MIN_VOLTS +
                ((Servo / (SERVO_MAX - SERVO_MIN)) *
                (SERVO_MAX_VOLTS - SERVO_MIN_VOLTS ));

    Double mv = new Double (MotorVolts);
    Double sv = new Double (ServoVolts);
    Integer motorint= new Integer((int)Motor);
    Integer servoint= new Integer((int)Servo);

    //Output motor values to control surface
    joyStickPanel.setText ("SPD:" + motorint.toString() + " STG:" +
                           servoint.toString());

    //Send control voltages
    InetAddress hostAdress= null;
```

```
    String motor_str = mv.toString() + "00000";
    String servo_str = sv.toString() + "00000";

    String cmdStr = motor_str.substring(0,5) + " " +
                    servo_str.substring(0,5) + "\n\n\n\n";
    byte[] cmdBytes = cmdStr.getBytes();
    try {
        DatagramPacket cmdPack = new DatagramPacket(cmdBytes,
                                 cmdBytes.length,
                                 hostAdress.getByName(ROBOT_IP_ADRESS),
                                 CMD_PORT);
        cmdSock.send(cmdPack);
        navModeField.setText("MANUAL");
    } catch (Exception ex) {
        messageArea.append("\nCannot Send Control Voltages.\n");
        return;
    }
}//sendControlData

/**
 * Increase current waypoint indicator by one
 */
public void increaseCurrentWaypoint() {
    currentWpt++;
    if (currentWpt > (MAX_WAYPOINTS - 1)) {
        currentWpt = (MAX_WAYPOINTS - 1);
    }
    wayPointField.setSelectedIndex(currentWpt);
}//increaseCurrentWaypoint

/**
 * Decrease current waypoint indicator by one
 */
public void decreaseCurrentWaypoint() {
    currentWpt--;
    if (currentWpt < 0) {
        currentWpt = 0;
    }
    wayPointField.setSelectedIndex(currentWpt);
}//decreaseCurrentWaypoint

/**
 * When displaying navigation map, draw WayPoint symbols and robot symbol.
 * Otherwise display camera image.
 */
public void draw() {
    if (videoMode == NAV_MAP) {
        //Update map
        File file = new File(NAVMAP_FILE_NAME);
        try {
            img = ImageIO.read(file);
        } catch (Exception ex) {
            messageArea.append("\nCannot Read Navigation Map.\n");
        }

        mapIcon = new ImageIcon(img);
        map.setIcon(mapIcon);
        mapGraph = img.getGraphics();
        //Draw Waypoints
        double lat, lon, screenX, screenY;
        for (int i = 0; i < MAX_WAYPOINTS; i++) {
            lat = route[i].getLatNum();
            lon = route[i].getLonNum();
```

```
            screenY = scalePos1y + ((lat - scalePos1Lat) / scaleY);
            screenX = scalePos1x + ((lon - scalePos1Lon) / scaleX);

            mapGraph.setColor(Color.RED);
            mapGraph.drawOval((int)(screenX - WptCircleDia / 2 - 1),
                              (int)(screenY - WptCircleDia / 2 - 1),
                              (int)WptCircleDia,
                              (int)WptCircleDia);
            Integer ii = new Integer(i + 1);
            mapGraph.drawString(ii.toString(),(int)(screenX +
                                WptCircleDia / 2), (int)screenY);
        }
        //Draw Robot Position & Heading
        lat = robotPos.getLatNum();
        lon = robotPos.getLonNum();

        double x1,y1,x2,y2;
        y1 = scalePos1y + ((lat - scalePos1Lat) / scaleY);
        x1 = scalePos1x + ((lon - scalePos1Lon) / scaleX);

        mapGraph.setColor(Color.BLUE);
        mapGraph.drawOval((int)(x1 - WptCircleDia / 2 - 1),
                          (int)(y1 - WptCircleDia / 2 - 1),
                          (int)WptCircleDia,
                          (int)WptCircleDia);
        //robotHeading = robotHeading % 360;
        double robotHeadingRad = (Math.toRadians(robotHeading)) -
                                 ((Math.PI)/2);
        y2 = y1 + ROBOT_HEADING_TICK_LENGTH * Math.sin(robotHeadingRad);
        x2 = x1 + ROBOT_HEADING_TICK_LENGTH * Math.cos(robotHeadingRad);
        mapGraph.drawLine((int)x1,(int)y1,(int)x2,(int)y2);

        map.repaint();
        img.flush();
        mapGraph.dispose();

    } else if (videoMode == VIS_CAM) {
        map.setIcon(null);
        byte[] camByteArray = new byte[34402];

        //-------------------------------------------------------------
        //This part is for test purpose only
        //Data is gonna taken directly from camStream
        try {
            FileInputStream cf = new FileInputStream("sample.dat");
            DataInputStream cd = new DataInputStream(cf);
            cf.read(camByteArray, 0, 34402);
            cd.close();
        } catch (Exception ex) {}
        //-------------------------------------------------------------

        //camByteArray = camStream.getBytes();

        int[] camArray = new int[camByteArray.length];

        for (int i = 0; i < camArray.length; i++) {
            camArray[i] = camByteArray[i] & 0xFF;
        }

        mapGraph = map.getGraphics();

        int R = 0;
```

```
        int G = 0;
        int B = 0;

        Color pixelColor = new Color(0, 0, 0);

        for (int counter = 0; counter < camArray.length; counter++) {
            if ((camArray[counter] >= 16) && (camArray[counter]
                    <= 240)) {
                if (rgb == 0) {
                    R = camArray[counter];
                    rgb++;
                    continue;
                } else if (rgb == 1) {
                    G = camArray[counter];
                    rgb++;
                    continue;
                } else {
                    B = camArray[counter];
                    int pix;

                    pix=0;
                    pix+=B;
                    pix+=G<<8;
                    pix+=R<<16;
                    pixelColor = new Color(pix);
                    mapGraph.setColor(pixelColor);

                    mapGraph.fillRect((int)Math.round(visCurX),
                                    (int)Math.round(visCurY),
                                    (int)Math.round(VIS_PIXEL_WIDTH),
                                    (int)Math.round(VIS_PIXEL_HEIGHT));
                    visCurY -= VIS_PIXEL_HEIGHT;
                    rgb = 0;
                    continue;
                }
            } else if (camArray[counter] == 2) {
                visCurX -= VIS_PIXEL_WIDTH;
                visCurY = VIS_ROWS * VIS_PIXEL_HEIGHT;
                rgb = 0;
                continue;
            } else if (camArray[counter] == 3) {
                visCurX = 1.5 * VIS_COLUMNS * VIS_PIXEL_WIDTH;
                visCurY = VIS_ROWS * VIS_PIXEL_HEIGHT;
                rgb = 0;
                break;
            } else if (camArray[counter] == 1){
                visCurX = VIS_COLUMNS * VIS_PIXEL_WIDTH;
                visCurY = VIS_ROWS * VIS_PIXEL_HEIGHT;
                rgb = 0;
                continue;
            } else {
                continue;
            }
        }
    } else {
        //IR CAM
        return;
    }
}//draw

/**
 * Get rid of leading, trailing and middle spaces
 */
```

```
public String cleanString(String s) {
    int spacePos;
    s = s.trim();
    while ((spacePos = s.indexOf(" ")) != -1 ) {
        s = s.substring(0, spacePos) +
            s.substring(spacePos + 1, s.length());
    }
    return s;
}//cleanString

/**
 * Listen for and handle mouse clicking events on GUI elements. This
 * happens when map is getting scaled or working with WayPoints. The
 * method discriminates the call by caller's Name and do not react to
 * other callers.
 *
 * @param e MouseEvent to be handled
 */
public void mouseClicked(MouseEvent e) {
    //Left Double Click on Map
    if ((e.getComponent().getName()).equals("M")) {
        if ((e.getButton() == 1) && (e.getClickCount() == 2)) {
            if (videoMode != NAV_MAP) {
                return;
            }
            //If WayPoint is being added
            if (nowScaling == 0) {
                WayPoint temp = new WayPoint();
                temp = nwpt.getWayPointData(this, mapCursorWpt);
                if (temp == null) {
                    return;
                }
                if (wptInsertMode == INSERT) {
                    for (int i=(MAX_WAYPOINTS - 1); i > currentWpt; i-- ) {
                        route[i] = route[i - 1];
                    }
                }
                route[currentWpt] =temp;
                draw();
                increaseCurrentWaypoint();
                return;
            }
            //If map scaling step 1 is happening
            if (nowScaling == 1) {
                scalingWpt1 = nwpt.getWayPointData(this, mapCursorWpt);
                if (scalingWpt1 == null) {
                    return;
                }
                nowScaling =2;
                scalingButton.setText("SCALING Step-2");
                scalePos1Lat = scalingWpt1.getLatNum();
                scalePos1Lon = scalingWpt1.getLonNum();
                scalePos1x = e.getX();
                scalePos1y = e.getY();
                return;
            }
            //If map scaling step 2 is happening
            if (nowScaling == 2) {
                scalingWpt2 = nwpt.getWayPointData(this, mapCursorWpt);
                if (scalingWpt2 == null) {
                    return;
                }
                nowScaling =0;
```

```
                scalingButton.setText("PERFORM MAP SCALING");
                messageArea.append("\nFollowing Points are Processed:\n");
                messageArea.append(scalingWpt1.getLatLon() + "\n");
                messageArea.append(scalingWpt2.getLatLon() + "\n");
                scalePos2Lat = scalingWpt2.getLatNum();
                scalePos2Lon = scalingWpt2.getLonNum();
                scalePos2x = e.getX();
                scalePos2y = e.getY();

                scaleX= (scalePos2Lon - scalePos1Lon) /
                        (scalePos2x - scalePos1x);
                scaleY= (scalePos2Lat - scalePos1Lat) /
                        (scalePos2y - scalePos1y);

                return;
            }
        }
    }
}//mouseClicked

//Events not being implemented
public void mousePressed(MouseEvent e) {
}
public void mouseReleased(MouseEvent e) {
}
public void mouseEntered(MouseEvent e) {
}
public void mouseExited(MouseEvent e) {
}

/**
 * Listen for and handle mouse dragging events on GUI elements. This
 * happens when manual control joystick is used. The method
 * discriminates the call by caller's Name and do not react to other
 * callers.
 *
 * @param e MouseEvent to be handled
 */
public void mouseDragged(MouseEvent e) {
    if ((e.getComponent().getName()).equals("J")) {
        Dimension rv = new Dimension();
        joyStickPanel.getSize(rv);
        double maxx = rv.getWidth() / 2;
        double maxy = rv.getHeight() / 2;

 /**
  *  x and y values from joystick normalized to 0 to 1 ????????????
  */

        double x = e.getX() / maxx - 1;
        double y = (maxy - e.getY()) / maxy;

        //Calculate motor values (not voltages)
        Motor = ((y) * ((MOTOR_MAX - MOTOR_MIN) / 2) + MOTOR_MIN);
        Servo = ((x) * ((SERVO_MAX - SERVO_MIN) / 2) + SERVO_MIN);

        //Check for overflows
        if (Motor > MOTOR_MAX) Motor = MOTOR_MAX;
        if (Motor < MOTOR_MIN) Motor = MOTOR_MIN;
        if (Servo > SERVO_MAX) Servo = SERVO_MAX;
        if (Servo < SERVO_MIN) Servo = SERVO_MIN;

        //Set current values not to send every time if nothing changed
```

```
        if ((Motor != MotorOld) || (Servo != ServoOld)) {
            sendControlData();
            MotorOld = Motor;
            ServoOld = Servo;
        }
    }
}//mouseDragged

/**
 * Listen for and handle mouse moving events on GUI elements. This
 * happens when cursor is moved over the map. This method computes and
 * outputs the current position. The method discriminates the call by
 * caller's Name and do not react to other callers.
 *
 * @param e MouseEvent to be handled
 */
public void mouseMoved(MouseEvent e) {
    if (videoMode != NAV_MAP) {
        return;
    }
    if ((e.getComponent().getName()).equals("M")) {
        double curX, curY;

        curX = scalePos1Lat + (((e.getY() - scalePos1y )) * scaleY);
        curY = scalePos1Lon + (((e.getX() - scalePos1x)) * scaleX);

        try {
            mapCursorWpt = new WayPoint(curX, curY, "Turn");
        } catch (Exception ex) {
            return;
        }
        cursorCoord.setText(mapCursorWpt.getLatLon());
    }
}//mouseMoved

/**
 * Perform tasks which should be done immediately the program starts
 * such as welcoming user, establishing connections and loading map
 * scaling data.
.*.
 * @param e WindowEvent to be handled
 */
public void windowOpened(WindowEvent e) {
    //Welcome user

    messageArea.append("Welcome to the Lopez Control Interface.\n");
    messageArea.append("Written by: Kubilay Uzun,\n");
    messageArea.append("James Knoll, Robert Williams.\n");
    messageArea.append("Modified by Jason Ward\n");
    messageArea.append("Naval Postgraduate School\n");
    messageArea.append("Monterey, California\n");

    //Establish Connections
    establishDatagramConnections();

    //Read Scaling Data
    try {
        fStorIn = new FileInputStream("Storage.dat");
        storIn = new DataInputStream(fStorIn);

        scalePos1Lat = storIn.readDouble();
        scalePos1Lon = storIn.readDouble();
        scalePos1y = storIn.readDouble();
```

```
        scalePos1x = storIn.readDouble();
        scaleY = storIn.readDouble();
        scaleX = storIn.readDouble();
    } catch (Exception ex) {
        messageArea.append("\nCannot Read Map Scaling Data.\n");
    }
}//windowOpened

/**
 * Perform tasks which should be done just before the program exits
 * such as closing sockets, saving map scaling data.
 *
 * @param e WindowEvent to be handled
 */
public void windowClosing(WindowEvent e) {
    //Close sockets
    try {
        cmdSock.close();
        routeSock.close();
    } catch (Exception ex) {
        messageArea.append("\nCannot Close Sockets.\n");
    }
    //Write Scaling Data
    try {
        fStorOut = new FileOutputStream("Storage.dat");
        storOut = new DataOutputStream(fStorOut);

        storOut.writeDouble(scalePos1Lat);
        storOut.writeDouble(scalePos1Lon);
        storOut.writeDouble(scalePos1y);
        storOut.writeDouble(scalePos1x);
        storOut.writeDouble(scaleY);
        storOut.writeDouble(scaleX);
    } catch (Exception ex) {
        messageArea.append("\nCannot Write Map Scaling Data.\n");
    }
}//windowClosing

//Events not being implemented
public void windowClosed(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}

/**
 * Perform numerous tasks including all functional buttons
 * and the timer
 *
 * @param ev ActionEvent to be handled
 */
public void actionPerformed(ActionEvent ev) {
    //Timer event happened
    if (ev.getActionCommand() == null) {
        boolean anythingChanged = false;

        //Query packet receiving threads
        String gpsStream = gpsThread.getReceivedData();
```

```
String compassStream = compassThread.getReceivedData();
String errorStream = errorThread.getReceivedData();


if (videoMode == VIS_CAM) {
    camStream = camThread.getReceivedData();
    if (camStream != null) {
        //anythingChanged = true; //original position
    }
    anythingChanged = true; //temporary position
}

//If GPS socket have something
if (gpsStream != null) {
    //System.out.println(gpsStream + "\n");
    //Tokenize GPS data
    StringTokenizer tok = new StringTokenizer(gpsStream, ",");
    if (tok.countTokens() < 8) {
        messageArea.append("\nCorrupted GPS Data.\n");
        return;
    }
    tok.nextToken(); //ignore header
    String GPSTime = tok.nextToken();
    GPSTime = cleanString(GPSTime);
    String GPSLatNums = tok.nextToken();
    GPSLatNums = cleanString(GPSLatNums);
    String GPSLatHemi = tok.nextToken();
    GPSLatHemi = cleanString(GPSLatHemi);
    String GPSLonNums = tok.nextToken();
    GPSLonNums = cleanString(GPSLonNums);
    String GPSLonHemi = tok.nextToken();
    GPSLonHemi = cleanString(GPSLonHemi);
    String GPSStatus = tok.nextToken();
    GPSStatus = cleanString(GPSStatus);
    String GPSSatellites = tok.nextToken();
    GPSSatellites = cleanString(GPSSatellites);

    GPSTime = GPSTime.substring(0,2) + ":" +
              GPSTime.substring(2,4) + ":" +
              GPSTime.substring(4,6);

    String GPSPos = GPSLatHemi +
                    GPSLatNums.substring(0,2) + " " +
                    GPSLatNums.substring(2,9) + " " +
                    GPSLonHemi +
                    GPSLonNums.substring(0,3) + " " +
                    GPSLonNums.substring(3,10);
    if (GPSStatus.equals("0")) {
        GPSStatus = "GPS not Available";
    } else if (GPSStatus.equals("1")) {
        GPSStatus = "GPS Available";
    } else {
        GPSStatus = "GPS Differential Fix";
    }

    fixTimeField.setText(GPSTime);
    numberOfSatellitesField.setText(GPSSatellites);
    gpsField.setText(GPSStatus);
    try {
        robotPos = new WayPoint(GPSPos, "Turn");
    } catch (Exception ex) {
        messageArea.append("\n Cannot Parse Robot Position.\n");
    }
```

```
            latField.setText(robotPos.getLat());
            lonField.setText(robotPos.getLon());
            anythingChanged = true;
        }

        //If Compass socket have something
        if (compassStream != null) {
            //System.out.println(compassStream + "\n");
            //Tokenize compass data
            StringTokenizer tok = new StringTokenizer(compassStream, ",");
            if (tok.countTokens() < 6) {
                messageArea.append("\nCorrupted Compass Data.\n");
                return;
            }
            tok.nextToken(); //ignore header
            String compassHeading = tok.nextToken();
            compassHeading = cleanString(compassHeading);
            tok.nextToken(); //ignore
            String compassRoll = tok.nextToken();
            compassRoll = cleanString(compassRoll);     //Roll and Pitch
            tok.nextToken(); //ignore                   //switched to deal
            String compassPitch = tok.nextToken();      //with compass
            compassPitch = cleanString(compassPitch);   //rotation

            float underlyingValue=Float.parseFloat(compassPitch);
            underlyingValue=-1*underlyingValue;
            compassPitch=Float.toString(underlyingValue);
            //The three lines above change the sign of the compass
            //pitch. They should be removed along with the parsing fix
            //above it if compass is rotated to be in line with robot body

            headField.setText(compassHeading);
            Double rh = new Double(compassHeading);
            try {
                robotHeading = rh.parseDouble(compassHeading);
            } catch(Exception ex) {
                messageArea.append("\nCannot Parse Compass Heading.\n");
            }
            pitchField.setText(compassPitch);
            rollField.setText(compassRoll);
            anythingChanged = true;
        }

        //If error socket have something
        if (errorStream != null) {
            if(!(errorStream.equals(oldErrorMessage))) {
                messageArea.append("\nLOPEZ: " +
                    errorStream.substring(1,errorStream.length()) + "\n" );
                oldErrorMessage = errorStream;
            }
        }

        if (anythingChanged) {
            draw();
        }
        return;
    }

    //Motor control event Happened
    boolean isMotorEvent = false;
    if (ev.getActionCommand().equals("STOP")) {
        Motor = (MOTOR_MAX + MOTOR_MIN) / 2;
        Servo = (SERVO_MAX + SERVO_MIN) / 2;
```

```
    isMotorEvent = true;
}else if (ev.getActionCommand().equals("FORWARD")) {
    Motor = MOTOR_MAX;
    Servo = (SERVO_MAX + SERVO_MIN) / 2;
    isMotorEvent = true;
}else if (ev.getActionCommand().equals("REVERSE")) {
    Motor = MOTOR_MIN;
    Servo = (SERVO_MAX + SERVO_MIN) / 2;
    isMotorEvent = true;
}else if (ev.getActionCommand().equals("RIGHT")) {
    Motor = MOTOR_MAX;
    Servo = SERVO_MIN;
    isMotorEvent = true;
}else if (ev.getActionCommand().equals("LEFT")) {
    Motor = MOTOR_MAX;
    Servo = SERVO_MAX;
    isMotorEvent = true;
} else if (ev.getActionCommand().equals("STOP")){
    Motor = (MOTOR_MAX + MOTOR_MIN) / 2;
    Servo = (SERVO_MAX + SERVO_MIN) / 2;
    isMotorEvent = true;
}

if (isMotorEvent) {
    if (((Motor != MotorOld) || (Servo != ServoOld)) ||
        (ev.getActionCommand().equals("STOP"))) {
        sendControlData();
        MotorOld = Motor;
        ServoOld = Servo;
    }
    return;
}

//Map Scaling event happened
if (ev.getActionCommand().equals("PERFORM MAP SCALING")) {
    if (videoMode != NAV_MAP) {
        return;
    }
    nowScaling = 1;
    JOptionPane.showMessageDialog(this, "Double Click and Enter 2 " +
                                  "Points on the map.");
    scalingButton.setText("SCALING Step-1");
    return;
}

//Increasing and decreasing current wpt event happened
if (ev.getActionCommand().equals(">")) {
    increaseCurrentWaypoint();
    return;
}
if (ev.getActionCommand().equals("<")) {
    decreaseCurrentWaypoint();
    return;
}
if (ev.getActionCommand().equals("comboBoxChanged")) {
    currentWpt = wayPointField.getSelectedIndex();
    return;
}

//Delete WayPoint event happened
if (ev.getActionCommand().equals("DELETE WAYPOINT")) {
    int pos, max;
    for (pos = currentWpt; pos < (MAX_WAYPOINTS - 1); pos++) {
```

```
            route[pos] = route[pos + 1];
        }
        route[MAX_WAYPOINTS - 1] = new WayPoint();
        draw();
        return;
    }

    //Insert/override event happened
    if (ev.getActionCommand().equals("Inserting")) {
        insertWpt.setText("Overriding");
        wptInsertMode = OVERRIDE;
        return;
    }
    if (ev.getActionCommand().equals("Overriding")) {
        insertWpt.setText("Inserting");
        wptInsertMode = INSERT;
        return;
    }

    //Edit WayPoint event happened
    if (ev.getActionCommand().equals("EDIT WAYPOINT")) {
        WayPoint temp = new WayPoint();
        temp = nwpt.getWayPointData(this, route[currentWpt]);
        if (temp == null) {
            return;
        }
        route[currentWpt] = temp;
        draw();
    }

    //Save/Load route event happened
    if (ev.getActionCommand().equals("SAVE ROUTE")) {
        FileDialog sr = new FileDialog(this, "SAVE ROUTE",
                                       FileDialog.SAVE);
        sr.show();
        String fn = sr.getFile();
        try {
            fRouteOut = new FileOutputStream(fn);
            routeOut = new ObjectOutputStream(fRouteOut);
            for (int i=0; i < MAX_WAYPOINTS; i++) {
                routeOut.writeObject(route[i]);
            }
            routeOut.close();
        } catch (Exception ex) {
            messageArea.append("\nCannot Save Route.\n");
        }
        draw();
        return;
    }
    if (ev.getActionCommand().equals("LOAD ROUTE")) {
        FileDialog sr = new FileDialog(this, "LOAD ROUTE",
                                       FileDialog.LOAD);
        sr.show();
        String fn = sr.getFile();
        try {
            fRouteIn = new FileInputStream(fn);
            routeIn = new ObjectInputStream(fRouteIn);
            for (int i=0; i < MAX_WAYPOINTS; i++) {
                route[i] = (WayPoint)routeIn.readObject();
            }
            routeIn.close();
        } catch (Exception ex) {
            messageArea.append("\nCannot Load Route.\n");
```

```
            }
            draw();
            return;
        }

        //Send route event happened
        if (ev.getActionCommand().equals("SEND ROUTE")) {
            String routeStream = "$" + route[0].getPack() + " ";

            for (int i =1; i < MAX_WAYPOINTS; i++ ) {
                routeStream = routeStream + route[i].getPack()+ " ";
            }
            routeStream = routeStream + "\n\n\n\n\n";

            InetAddress hostAdress= null;
            //send route
            byte[] routeBytes = routeStream.getBytes();
            try {
                DatagramPacket routePack = new DatagramPacket(routeBytes,
                                           routeBytes.length,
                                           hostAdress.getByName(ROBOT_IP_ADRESS),
                                           ROUTE_PORT);
                routeSock.send(routePack);
                navModeField.setText("AUTO");
            } catch (Exception ex) {
                messageArea.append("\nCannot Send Route.\n");
                return;
            }
        }

        if (ev.getActionCommand().equals("NAVIGATION MAP")) {
            videoMode = NAV_MAP;
            draw();
            return;
        }

        if (ev.getActionCommand().equals("VISUAL CAM")) {
            videoMode = VIS_CAM;
            draw();
            return;
        }

    }//Action events

    /**
     * Main. Set the GUI frame visible only.
     *
     * @param args The string array which is arguments passed
     * @exception Exception
     */
    public static void main(String[] args) throws Exception {
        MainApp ma = new MainApp();
        ma.setVisible(true);
    }//main
}//MainApp
```

LIST OF REFERENCES

1. Quinn, R.D., Nelson, G.M., Ritzmann, R.E., Bachmann, R.J., Kingsley, D.A., Offi, J.T. and Allen, T.J. “Parallel Strategies for Implementing Biological Principles Into Mobile Robots”, Int Journal of Robotics Research, Vol.22(3) pp 169-186, 2003.

2. Watson, J.T., Ritzmann, R.E., Zill, S.N., Pollack, A.J., “Control of obstacle climbing in the Cockroach, Blaberus discoidalis: I. Kinematics” Journal of Comp Physiology Vol 188, pp 39-53, 2002.

3. Lambrecht, Bram G. A., “A Small, Insect Inpsired Robot that Runs and Jumps”, Masters Thesis, Case Western Reserve University, January 2005.

4. Allen, Thomas, J., “Abstracted Biological Principles and Reduced Actuation Applied to the Design of a Walking Vehicle”, Masters Thesis, Case Western Reserve University, August 2004.

5. Boxerbaum, A.S., Werk, Philip, Quinn, R.D., Vaidyanathan, Ravi, “Design of an Autonomous Amphibious Robot for Surf Zone Operation: Part I Mechanical Design for Multi-Mode Mobility”, Masters Thesis, Case Western Reserve University, March 2005.

6. Chicoine, Andrew G., “The Naval Postgraduate School’s Small Robotics Technology Initiative: Initial Platform Integration and Testing.”, Masters Thesis, Naval Postgraduate School, December 2001.

7. Novak Super Rooster, [http://www.teamnovak.com/products/esc/super_rooster/index.html] May 2005.

8. Apogee Lithium Polymer Battery Pack, [http://www.robotcombat.com/marketplace_lipoly-ap.html] May 2005.

9. Z-world Products BL2000, [http://www.zworld.com/products/bl2000/largeView.shtml] May 2005.

10. Garmin Global Positioning Systems-Products: OEM/Sensors [http://www.garmin.com/products/gps16/] May 2005.

11. Honeywell Magnetic Sensors Datasheets, [http://www.ssec.honeywell.com/magnetic/datasheets/hmr3000.pdf] May 2005.

12. Crossbow Industries LP Series Accelerometer Datasheets, [http://www.xbow.com/Products/Product_pdf_files/Accel_pdf/LP_Series_Datasheet.pdf] May 2005.

13. Proxim RangeLan2 Ethernet Adapter, [http://support.proxim.com/cgi-bin/proxim.cfg/php/enduser/std_adp.php?p_faqid=440&p_created=1068662148&p_sid=cGQxNeGh&p_lva=&p_sp=cF9zcmNoPTEmcF9zb3J0X2J5PTI6MiZwX2dyaWRzb3J0PSZwX3Jvd19jbnQ9MSZwX3Byb2RzPTAmcF9jYXRzPTAmcF9wdj0mcF9jdj0mcF9zZWFyY2hfdHlwZT1hbnN3ZXJzLnNlYXJjaF9ubCZwX3BhZ2U9MSZwX3NlYXJjaF90ZXh0PVJhbmdlTEFOMiA3OTIx&p_li=&p_topview=1] (Password Protected) May 2005.

14. Uzan, Kubilay, et.al. SE4015 Course Presentation, Naval Postgraduate School, September 2003.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Physics Department
   Naval Postgraduate School
   Monterey, California

4. Dick Harkins
   Department of Applied Physics
   Naval Postgraduate School
   Monterey, California

5. Dr. Ravi Vaidyanathan
   Department of Systems Engineering
   Naval Postgraduate School
   Monterey, California

6. Chuck Bernstein
   Office of Naval Research
   Arlington, VA

7. Tom Swean, Jr.
   Ocean Engr & Marine Systems
   Office of Naval Research
   Arlington, Virginia

8. Z-World, Inc.
   Davis, California

9. Bill BirmingHam
   PFM Distribution, Inc.
   Belleville, Illinois